CENTRO UNIVERSITÁRIO DINÂMICA DAS CATARATAS

FERNANDO DEMARCHI NATIVIDADE LUIZ

UTILIZAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA A PREDIÇÃO EM BOLSAS DE VALORES ATRAVÉS DE SÉRIES TEMPORAIS

FOZ DO IGUAÇU

2017

FERNANDO DEMARCHI NATIVIDADE LUIZ

UTILIZAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA A PREDIÇÃO EM BOLSAS DE VALORES ATRAVÉS DE SÉRIES TEMPORAIS

Trabalho de conclusão de curso apresentado como requisito obrigatório para obtenção do título de Bacharel em Ciência da Computaçãodo Centro Universitário Dinâmica das Cataratas.

Orientador: Prof. Me. Valmei Abreu Júnior

Coorientador: Prof. Dr. Miguel Diógenes Matrakas

FOZ DO IGUAÇU

2017

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS

API	*Application Programming Interface* - Interface de programação de aplicações
BSD	*Berkeley Software Distribution*
EAM	Erro Absoluto Médio
EQM	Erro quadrático médio
IDE	*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado
MACD	*Moving Average Convergence Divergence* - Convergência e Divergência de Médias Móveis
MLP	*Multilayer Perceptron* - Perceptron de Múltiplas Camadas
NASDAQ	*National Association of Securities Dealers Automated Quotation System*
PDF	*Portable Document Format* - Formato Portátil de Documento
PyBrain	*Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library* - Reforço de aprendizagem baseada em Python, Biblioteca de Inteligência Artificial e Rede Neural
REQM	Raiz do Erro Quadrático Médio
SQL	*Structured Query Language* - Linguagem de Consulta Estruturada
SVM	*Support-Vector-Machines* - Máquinas de Vetores de Suporte
XML	*eXtensible Markup Language* - Linguagem Extensível de Marcação Genérica

SUMÁRIO

1 INTRODUÇÃO

A informatização do mercado acionário, que permite a movimentação de compra e venda de ações de forma eletrônica e automática, tornou-se fundamental ao longo das últimas décadas, gerando uma série de mudanças na forma em que as negociações são realizadas, se comparado ao modelo de negociação anterior, onde as movimentações aconteciam em uma unidade central, com a presença física dos investidores. Portanto, esse novo modelo mudou a forma de atuação nas bolsas de valores, possibilitando a movimentação de capitais *online*, facilitando e aumentando a gama de possibilidades para a aquisição de novos acionistas. Atualmente, os papéis negociados mais relevantes em bolsas de valores, vêm de grandes empresas que dominam o mercado em suas respectivas áreas, gerando uma alta movimentação financeira que pode conceber um impacto tanto positivo como negativo no setor econômico, influenciando em decisões políticas e sociais que afetam não só a esfera econômica, mas também todos os níveis da sociedade (SHILLER, 2005).

Em contrapartida, houve, automaticamente, um impacto computacional significativo sobre esse novo paradigma do mercado de ações. Com o advento da Internet e o fácil acesso aos dados históricos dos papéis negociados nas bolsas de valores, através de *sites* que disponibilizam essas informações, técnicas computacionais têm se tornado uma grande aliada para o diagnóstico dos preços de ações e índices, aumentando significamente as pesquisas e os modelos computacionais que possam auxiliar no aperfeiçoamento e análise do mercado financeiro.

A complexidade em prever valores futuros no mercado de ações, é um estímulo consideravelmente grande para a comunidade científica internacional, tendo em vista a diminuição do risco de investimento em ativos financeiros. Nesse contexto, técnicas como aprendizado de máquina, Redes Neurais Artificiais e outras soluções de inteligência artificial, vêm sendo aplicadas arduamente para possibilitar um aumento significativo na predição de valores no mercado acionário (GAMBOGI, 2013).

Dessa maneira, uma abordagem que vem ganhando destaque, dentre as pesquisas referentes à predição de cotações futuras que utilizam séries temporais, são as Redes Neurais Artificiais (RNAs). A capacidade das RNAs em trabalhar com uma quantidade significativa de variáveis simultâneas, além da composição de sua estrutura maciçamente paralela e distribuída, evidenciam sua alta escala de poder computacional, concedendo-as habilidades de aprendizado e generalização de funções. Estas duas capacidades de processamento de informação, tornam possíveis para as RNAs resolver problemas de grande escala que, usando o processamento digital convencional, são consideradas computacionalmente inviáveis e intratáveis (ELPINIKI; KATARZYNA, 2016; HAYKIN, 2000).

Neste sentido, dada a importância das RNAs na predição de cotações futuras

de ações em bolsas de valores, o presente trabalho apresenta um estudo da aplicação destas RNAs na previsão de preços futuros dos mais relevantes papéis da NASDAQ *Stock Market*, o segundo maior mercado de ações do mundo, palco onde as maiores empresas de tecnologias investem e controlam seus ativos financeiros.

1.1 JUSTIFICATIVA

São vários os aspectos que impulsionam a necessidade de previsão dos índices acionários no mercado de investimentos. De fato, faz-se o principal motivo, garantir maior lucratividade possível com a realização de bons investimento, fazendo com que os investidores obtenham sempre uma vantagem sobre os demais que atuam no mercado financeiro.

Outro fator motivacional para a realização desta predição, utilizando RNAs, se dá pela sua capacidade de adaptação a mudanças bruscas e repentinas nos dados a serem analisados, sem que se afete a acuracidade de seus resultados, por não se tratar de um modelo pré-definido (MARANGONI, 2010).

O crescimento da utilização de RNAs aplicadas a importantes áreas do contexto social, é mais um fator motivacional para sua utilização. Além da aplicação na área financeira apresentada neste trabalho, ferramentas tecnológicas que implementam esta inteligência computacional têm-se feito úteis em áreas como: recursos humanos, marketing, medicina, engenharia, dentre outras. Na área da medicina, por exemplo, é válido citar sua utilidade no diagnóstico e, até mesmo, na prevenção de futuras doenças e patologias (MARANGONI, 2010).

Portanto, a variação dos preços das ações proporcionada por oscilações diárias, é uma excelente diretriz para o aperfeiçoamento de análises referente à inteligência computacional, dispondo de um ambiente propício para a aplicabilidade sobre séries temporais, através do acesso a uma quantidade contínua de dados que viabiliza criar e aprimorar diversos padrões de RNAs (GAMBOGI, 2013).

1.2 OBJETIVOS

A partir do tema definido, estabelecem-se os objetivos a serem alcançados ao término deste trabalho. Assim temos, respectivamente, o objetivo principal e os objetivos específicos, apresentados em ordem lógica de desenvolvimento.

1.2.1 Objetivo Geral

Atribuiu-se, como objetivo principal deste trabalho, especificar e aplicar um modelo de RNA para análise e predição de valores acionários através de séries temporais.

1.2.2 Objetivos Específicos

Com o objetivo geral determinado, tornaram-se necessários as definições de alguns objetivos específicos, sendo:

- Especificar uma arquitetura de RNA;
- Definir um ambiente para o desenvolvimento da RNA;
- Utilizar uma Interface de programação de aplicações (API) para a coleta dos dados que serão analisados;
- Treinar o modelo de RNA especificado;
- Realizar testes com o modelo treinado, através das ações utilizadas;
- Analisar os resultados obtidos pela implementação, comparando-os com os seus resultados reais e avaliando sua capacidade de precisão.

1.3 ESTRUTURA DO DOCUMENTO

Este documento será organizado em sete capítulos: introdução, revisão bibliográfica, fundamentação teórica, materiais e métodos, implementação, resultados, conclusões e referências bibliográficas.

O presente capítulo contextualiza e introduz o assunto a ser tratado, bem como define os objetivos a serem alcançados.

No segundo capítulo, são apresentados os trabalhos que foram utilizados como referência e motivação, além de demonstrar suas contribuições científicas.

A fundamentação teórica apresentada no terceiro capítulo, descreve o embasamento referente aos conceitos das técnicas que serão utilizadas para a realização deste estudo.

O quarto capítulo é destinado ao detalhamento de métodos, técnicas e procedimentos que têm por objetivo gerar conhecimento para viabilizar um melhor ambiente de desenvolvimento para o trabalho.

O quinto capítulo detalha a implementação dos métodos, que segue as especificações das técnicas levantadas no capítulo anterior.

No sexto capítulo são expostos os experimentos realizados sobre o modelo implementado, destacando sua capacidade de precisão aplicada ao objetivo do trabalho.

Por fim, no sétimo capítulo são tratadas as conclusões obtidas pela técnica de RNA desenvolvida, destacando suas vantagens e desvantagens. Também serão idealizadas

alternativas para trabalhos futuros, os quais dariam continuidade e complementariam a pesquisa realizada.

2 REVISÃO BIBLIOGRÁFICA

Várias técnicas conjugadas à inteligência artificial foram desenvolvidas e vêm sendo aperfeiçoadas ao longo do tempo, com o objetivo de possibilitar predições de boa acuracidade, tal como o estudo realizado por Clements e Hendry (1998) que trabalharam com predições utilizando regressão linear. Dentro dessa linha, também estão presentes as pesquisas que se basearam na utilização de RNAs, que são citadas como referência para analisar a oscilação do mercado acionário, como os estudos realizados por Faria et al. (2009), Kara, Boyacioglu e Baykan (2011) e White (1988), além de trabalhos desenvolvidos utilizando algoritmos genéticos, como apresentado no trabalho de Nayak, Misra e Behera (2012).

O primeiro modelo para previsão de preços no mercado de ações baseado em RNA, foi desenvolvido por White (1988), o autor utilizou um padrão de rede recorrente, denominado Feedforward, com o objetivo de analisar os retornos diários das ações da International Business Machines (IBM), para testar a teoria do mercado eficiente, proposta por Fama (1970), retratando que a oscilação dos preços das ações seguem uma tendência aleatória. Com bons resultados obtidos através da implementação da RNA, desde então, foi potencializada a quantidade de pesquisas referente à técnica utilizada.

Um dos trabalhos pioneiros em predição no mercado acionário com o auxílio de RNAs, foi o proposto por Kamijo e Tanigawa (1990), que utilizaram RNAs recorrentes para o reconhecimento de padrões, através dos gráficos gerados pela análise técnica da bolsa de Tóquio. O objetivo da pesquisa foi encontrar padrões classificados como possíveis indicadores de tendências para o investimento em um determinado ativo.

Trabalhos posteriores, como o realizado por Ding, Canu e Denoeux (1995), define a não existência de um paradigma adequado para predições com o uso de RNAs, necessitando da análise e identificação do problema para especificar o melhor modelo de arquitetura a ser aplicado. Ding, Canu e Denoeux (1995) também evidenciam a complexidade em especificar parâmetros relevantes para a fase de treinamento de uma RNA, pois o mesmo impacta diretamente no resultado obtido, tornando-se uma etapa de grande relevância na construção de um modelo estável.

Em seu estudo Guresen, Kayakutlu e Daim (2011) descreveram a utilização das RNAs como uma das melhores técnicas para modelar o mercado de ações, porque as mesmas podem ser facilmente adaptadas às mudanças do mercado acionário. Em seu trabalho, os autores utilizam uma rede Multilayer Perceptron (MLP) com um total de 80 neurônios, utilizando 20 para a camada de entrada, 40 para a camada oculta, e 20 para a camada de saída, aplicando o algoritmo Backpropagation como forma de treinamento, empregada para prever os valores dos índices das bolsas de valores americanas.

A utilização de RNAs em séries temporais, tem como referência o trabalho realizado por Waibel (1998) que evidencia duas abordagens para análise de séries temporais: Dimensionamento temporal e RNAs recorrentes. Em sua pesquisa, foi abordada a capacidade de generalização da RNA sobre um ambiente desconhecido, podendo auxiliar na resolução dos problemas que contam com diversos padrões distintos e com alta complexidade computacional. Waibel (1998) utilizou uma rede Feedforward para testar predições que obtivessem resultados relevantes e satisfatórios, utilizando um padrão de previsão denominado One Step Ahead. Este padrão trabalha, segundo o autor, com uma distribuição de N variáveis de entradas, através de uma alimentação iterativa até o momento T, onde a unidade de saída é representada por T + 1.

Zhang, Patuwo e HU (1988) também destacam modelos de previsões através de RNAs. O trabalho enfatizou a realização de múltiplas previsões futuras, denominado Multiple Step Ahead. Os autores descrevem dois métodos apresentados na literatura para este padrão, previsão iterativa e método direto. No primeiro método, os valores de predição são usados iterativamente, como material para previsões futuras. Neste caso, apenas um neurônio de saída é necessário. O segundo método, consiste em colocar várias saídas na RNA, correspondente ao resultado esperado da previsão, tendo maior relevância e utilidade para ambientes específicos e controlados.

Trabalhos recentes utilizando métodos híbridos também vêm sendo dedicados para melhorar a capacidade de modelagem sobre séries temporais. Na pesquisa produzida por Khandelwal, Ratnadip e Verma (2015) foi utilizado uma abordagem que cruza características do modelo matemático ARIMA com o de uma RNA. Os autores testaram os métodos ARIMA e RNA para separar componentes lineares e não-lineares, respectivamente, de séries temporais em valores econômicos, com o objetivo de encontrar um melhor padrão para tratar a oscilação dos dados obtidos.

Outra abordagem híbrida conhecida na literatura por ter resultados significativos juntamente com as RNAs, são os Mapas Cognitivos Fuzzy (FCMs). No trabalho realizado por Lu et al. (2014) por exemplo, é demonstrado o uso de um FCM trabalhando como um modelo de RNA recorrente, capacitando o protótipo a operar através de aprendizado, integrando assim as características de ambas as técnicas aplicadas.

É importante salientar que os trabalhos citados neste capítulo, por alguma peculiaridade ou detalhe técnico serviu como motivação para a idealização deste estudo, não podendo então deixar de citá-los. Além disso, provaram ser referência dentro de suas respectivas problemáticas, trazendo análises e conclusões significativas, que são essenciais para o aperfeiçoamento acadêmico e científico.

Portanto é almejado nos capítulos posteriores, adquirir conhecimento teórico suficiente que possibilite a realização de um trabalho que traga uma contribuição científica relevante, e que sirva como referência para os estudos futuros, como os aqui citados.

3 FUNDAMENTAÇÃO TEÓRICA

Nos tópicos que seguem, serão apresentados conceitos teóricos sobre o tema proposto, ou seja, será realizada uma contextualização desde o processo de funcionamento de RNAs até sua aplicabilidade no mercado acionário.

3.1 FUNDAMENTOS DE REDES NEURAIS ARTIFICIAIS

As RNAs procuram simular métodos de aprendizado do cérebro humano, através do uso de neurônios interligados. Uma RNA é inspirada nos neurônios biológicos e nos sistemas nervosos, logo, para entender o funcionamento da mesma, é necessário, primeiramente, entender o funcionamento dos neurônios biológicos (MENDONÇA NETO, 2014).

A partir deste conceito, uma RNA pode ser definida como uma estrutura interligada complexa de processamento, composta por neurônios e conexões (FERREIRA, 2004).

3.1.1 Neurônios Biológicos

Os neurônios são células, que desempenham o papel de conduzir os impulsos nervosos. Estas células especializadas são, portanto, as unidades básicas do sistema que processa as informações e estímulos de um cérebro(LENT, 2001).

A Figura 1 representa a estrutura básica de um neurônio biológico. Conceituando os componentes do neurônio, os dendritos são responsáveis pela recepção das informações, possuindo as extremidades ramificadas. O corpo celular é responsável pela integração das informações e os axônios são responsáveis pelo transporte dos impulsos nervosos de um neurônio para outro ou de um neurônio para uma glândula ou fibra muscular (LENT, 2001).

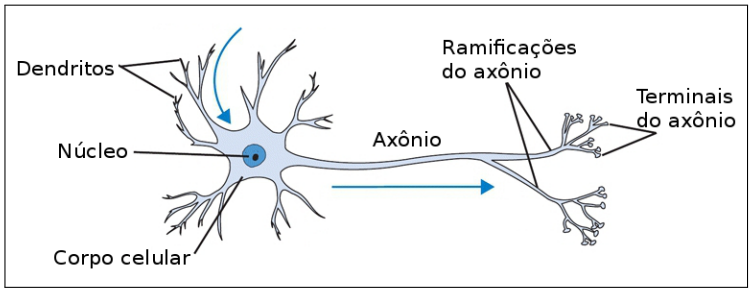


FIGURA 1 – Representação de um neurônio biológico
FONTE: Adaptado de Stanford (2014)

Cada dendrito possui em torno de mil a dez mil ramificações em suas extremidades. A transmissão de impulsos nervosos por meio da comunicação dos axônios com os

dendritos de neurônios adjacentes é chamada de sinapse, por sua vez, tal conjunto forma o sistema nervoso (MENDONÇA NETO, 2014).

3.1.2 Neurônios Artificiais

A estrutura de um neurônio artificial segue os mesmos conceitos relacionados aos neurônios biológicos, buscando realizar as mesmas funções, utilizando-se de conceitos matemáticos, aritméticos e de tecnologias computacionais.

A Figura 2 esboça a estrutura que compõe um neurônio artificial não-linear, demarcando cada item que o compõe.

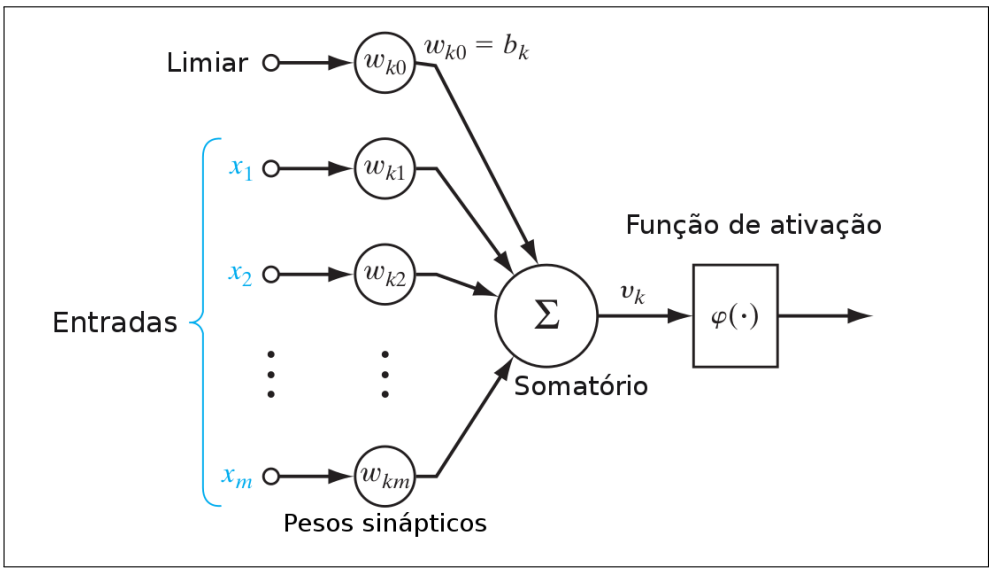


FIGURA 2 – Representação de um neurônio artificial
FONTE: Adaptado de Haykin (2009)

Analisando a figura da esquerda para a direita, pode-se observar que:

1. *O*s sinais de entrada x_i serão processados pela sinapse wk_i, conectadas ao neurônio;
2. *A*s sinapses ou elos de conexão, com seus respectivos pesos sinápticos que se multiplicarão ao sinal de entrada θ_i. Segundo Haykin (2000), ao contrário a uma sinapse do cérebro, o peso sináptico de um neurônio artificial pode estar em um intervalo que inclui tanto valores positivos quanto negativos;
3. *O* somatório $\sum$, que resulta na soma dos sinais de entrada x_i, ponderados pelas respectivas sinapses do neurônio;
4. *A* função de ativação, responsável por delimitar os resultados de saída possíveis a um intervalo finito, recebe como entrada o resultado v_k em $\varphi(.)$ para gerar o resultado.

3.1.3 Redes Neurais Artificiais

Uma RNA pode ser definida como uma ferramenta computacional, produzida e programada para realizar análise de dados, tal como seu comportamento e relações, visando trabalhar com os mesmos conceitos de um sistema nervoso, simulando o comportamento de um conjunto de neurônios biológicos. Por meio de complexos processos de treinamento e aprendizagem, as RNAs visam realizar o reconhecimento de padrões de dados, para que possa prever resultados, através de classificação ou generalização dos dados em questão (HAYKIN, 2009).

A representação de uma RNA pode ser visualizada na Figura 3, onde os nodos do grafo representam os neurônios, as arestas representam as conexões e as setas representam a direção do fluxo de sinal.

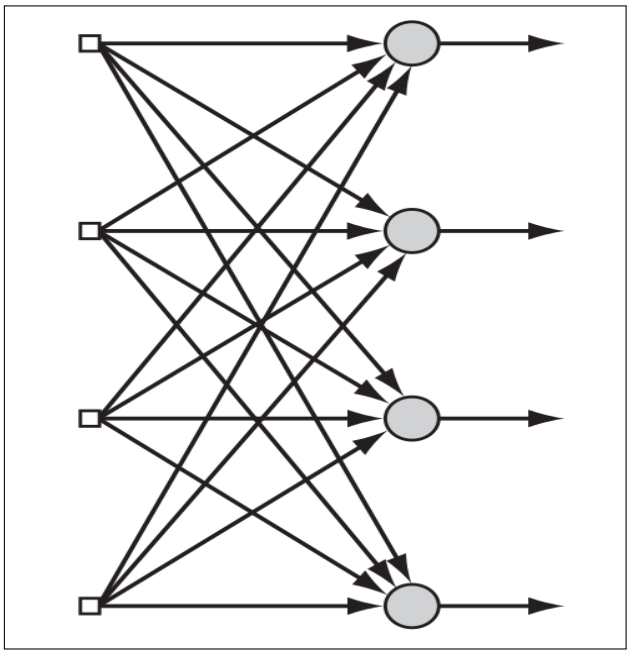


FIGURA 3 – Representação gráfica de uma Rede Neural Artificial
FONTE: Haykin (2009)

3.2 ARQUITETURA DE REDES NEURAIS ARTIFICIAIS

A forma como agrupam-se os neurônios artificiais em uma RNA é uma das principais características que definem seu tipo de arquitetura. Estes agrupamentos são baseados na forma como os neurônios são conectados no cérebro humano, de forma que as informações possam ser processadas dinamicamente ou iterativamente.

Biologicamente, as RNAs são organizadas e construídas de forma tridimensional por componentes microscópicos. Há uma forte restrição no número de camadas que a rede pode conter, limitando consideravelmente o tipo e o escopo da implementação da mesma, dependendo da complexidade do problema (HAYKIN, 2009).

A arquitetura de uma RNA pode ser classificada pela quantia de camadas ocultas e pelo sentido do fluxo de dados. Em relação ao número de camadas ocultas, ela pode ser de camada única (*single-layer*) ou multicamadas (*multilayer*). Quanto ao sentido do fluxo de dados, ela pode ser alimentada adiante (*feedforward*) ou recorrente (*feedback*) (MENDONÇA NETO, 2014).

As redes que possuem uma única camada são as que possuem um nó entre sua entrada e saída. Este tipo de rede é indicada para a solução de problemas linearmente separáveis. Já as que possuem multicamadas apresentam uma ou mais camadas entre seu início e fim. Essas camadas são chamadas de camadas escondidas (*hidden*, intermediárias ou ocultas).

A topologia de RNA mais popular atualmente é a rede direta com neurônios estáticos (*feedforward*) juntamente com a perceptron de múltiplas camadas (*Multilayer Perceptron*) que utiliza o algoritmo de retropropagação (*backpropagation*) como forma de treinamento (MENDONÇA NETO, 2014).

3.2.1 Redes Alimentadas Adiante com Camada Única

A organização de uma RNA em camadas adiante pode ser representada de uma maneira simples, onde os neurônios, distribuídos em camadas, recebem sinais de entrada que são projetados sobre os mesmos. Porém, nesta distribuição não ocorre o contrário, ou seja, o fluxo de alimentação ocorre sempre no sentido adiante ou acíclico.

Segundo Haykin (2000), na estrutura de camada única, a representação dos nós de neurônios responsáveis por receber e processar os sinais de entrada é dada em uma única camada. Para melhor entendimento deste tipo de estrutura, a Figura 4 exemplifica um modelo de RNA com essa característica.

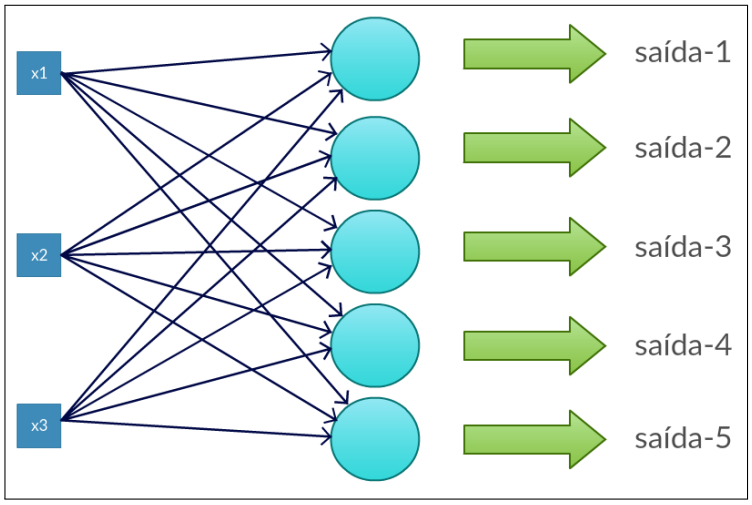


FIGURA 4 – Representação de uma Rede Neural Artificial com camada única
FONTE: Elaborado pelo autor

Para exemplificar este modelo de camada única, pode-se citar a RNA *Perceptron*

simples, criada por Frank Rosenblatt em 1957 nos laboratórios das forças militares. A classificação dos resultados de uma RNA desta categoria é ilustrada na Figura 5, onde seus resultados são linearmente separáveis, ou seja, este modelo de RNA é capaz de classificar de forma satisfatória, resultados que podem ser separados por uma reta ou hiperplano como fronteira de decisão (HAYKIN, 2009).

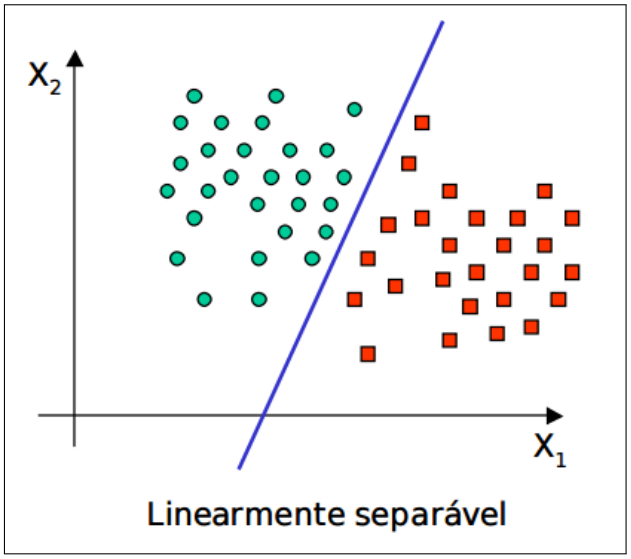


FIGURA 5 – Representação de resultados linearmente separáveis
FONTE: Oliveira (2002)

3.2.2 Redes Alimentadas Adiante com Múltiplas Camadas

A estrutura de múltiplas camadas, difere-se da primeira classe alimentada adiante em relação a composição de suas camadas. Esta classe de redes surgiu da necessidade em aperfeiçoar a capacidade de mapeamento que um modelo de camada única não proporciona. Um exemplo clássico é a função ou-exclusivo (XOR) (HAYKIN, 2009).

Tipicamente, a rede consiste em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. O sinal de entrada se propaga para frente através da rede, camada por camada. Este tipo de rede é bastante popular devido aos métodos de aprendizado bem distribuídos e de fácil uso (HAYKIN, 2000).

Sua representação pode ser vista na Figura 6, onde pode-se observar a presença de duas camadas intermediárias até a apresentação de seu resultado.

De acordo com Haykin (2000), qualquer rede semelhante à representada na Figura 6, é dita totalmente conectada, visto que todas as entradas x_i se conectam a todos os nós da camada oculta a_i. Caso a premissa não fosse estabelecida a rede seria dita parcialmente conectada.

Outro ponto que a difere da classe com camada única, é a possibilidade de trabalhar com problemas não-lineares, ou seja, os resultados não necessariamente são classificados de

forma satisfatória através apenas de uma reta ou hiperplano, o que aumenta a capacidade de RNAs com esta característica em resolver problemas mais complexos.

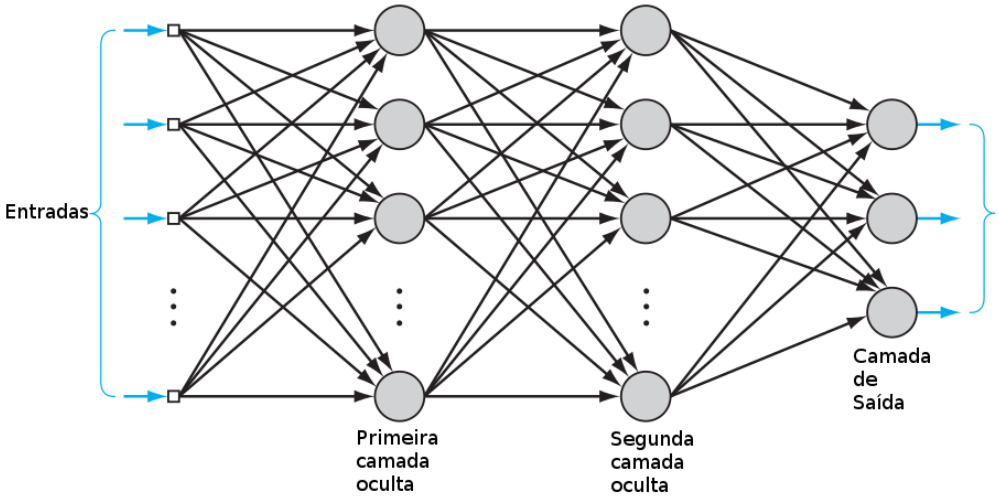


FIGURA 6 – Rede Neural Artificial com múltiplas camadas
FONTE: Adaptado de Haykin (2009)

A Figura 7 exemplifica a representação de um resultado não-linear proporcionado por uma rede de múltiplas camadas.

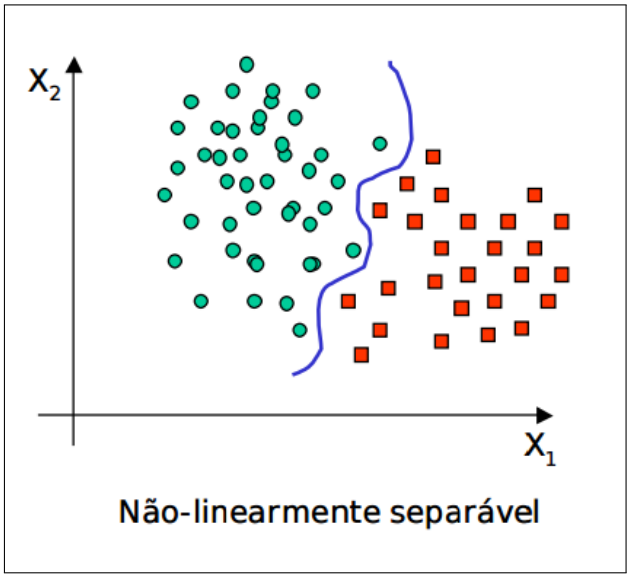


FIGURA 7 – Classificação de resultados não-lineares
FONTE: Oliveira (2002)

3.2.3 Redes de Camadas Recorrentes

Diferentemente da arquitetura de camadas simples adiante, a arquitetura de camadas recorrentes apresenta ao menos um laço de realimentação. A presença de realimentação de informação permite a criação de representações internas e dispositivos de memória capazes de processar e armazenar informações temporais e sinais sequenciais.

Em seu trabalho, Haykin (2000) cita um exemplo clássico de uma rede de camadas recorrentes, onde os neurônios em camada única alimentam seus sinais de saída novamente para as entradas de todos os neurônios.

A arquitetura de uma RNA recorrente pode ser observada na Figura 8, onde as entradas são realimentadas através da saída. Após realizado esse processamento, a rede apresenta o resultado calculado.

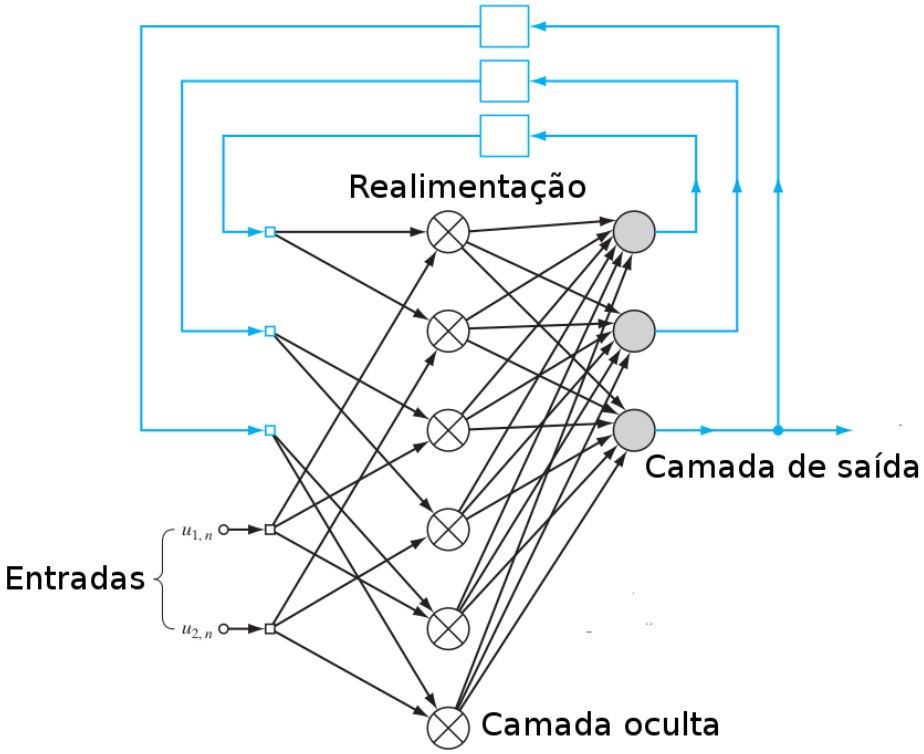


FIGURA 8 – Rede Neural Artificial recorrente
FONTE: Adaptado de Haykin (2009)

3.2.4 Redes de Hopfield

Dentre outros modelos e arquiteturas de redes que existem, pode-se citar Hopfield, desenvolvida pelo físico John Hopfield. Este modelo se caracteriza por ser do tipo *feedback*, por este motivo, estas redes dificilmente chegam a um estado instável, pois chegará um momento, após seu treinamento, em que o valor de saída para determinadas entradas se padronizam, alcançando sua estabilidade. O modelo utilizou uma função de energia como ferramenta para desenhar redes recorrentes e entender como funciona seu comportamento dinâmico. Desta forma, popularizou o uso desta arquitetura como uma memória associativa e para resolver problemas de otimização (CARDON; MULLER, 1994).

O modelo aplica um princípio chamado de armazenamento de informação como atratores dinamicamente estáveis. Para recuperar informações, utilizou um processo dinâmico de atualização dos estados dos neurônios, sendo que, o neurônio a ser atualizado,

é escolhido aleatoriamente. Dois modelos de redes foram apresentados por Hopfield: o analógico e o binário (SILVA, 2003).

3.2.5 Redes Auto-Organizáveis

Segundo Haykin (2009), Redes Auto-Organizáveis são modelos de 2 camadas que aceitam padrões de N-dimensões como entrada e os mapeia para um conjunto de neurônios de saída, o qual representa o espaço dos dados a serem agrupados. O mapa (camada) de saída, que é tipicamente bi-dimensional, representa as posições dos neurônios em relação aos seus vizinhos. A ideia é que neurônios topologicamente próximos respondam de maneira semelhante a entradas semelhantes. Para isso todos neurônios da camada de entrada são todos conectados aos neurônios de saída.

3.3 FUNÇÃO DE ATIVAÇÃO

O processamento em cada neurônio se dá através da função de ativação. A escolha da função de ativação de uma RNA é um processo de grande relevância, uma vez que esta função define como devem ser tratados seus dados de entrada. As funções de ativação podem ser classificadas como lineares ou não lineares (HAYKIN, 2000).

Segundo Haykin (2000), caracterizando-se pela limitação de valores de saída a um intervalo finito, destacam-se os modelos de funções: limiar, sigmóide e tangente hiperbólica.

3.3.1 Função Limiar

Para este exemplo de função, o seguinte cálculo deve ser utilizado:

$$y = \sum_{j=1}^{w} w_{kj}\ x_j + b_{k'}, \tag{3.1}$$

onde x é a entrada induzida no neurônio, gerando um resultado y_k dado por:

$$y_k = 1\ se\ x_j > 0 \tag{3.2}$$

ou

$$y_k = 0\ se\ x_j < 0 \tag{3.3}$$

Neste modelo a saída do neurônio assume o valor 1 se o campo local induzido daquele neurônio for positivo, e 0 se for negativo. Utilizam-se dos mesmos preceitos de regressão linear e de outras técnicas, para estimar resultados a partir de valores conhecidos.

3.3.2 Função Sigmóide

A função sigmóide, representada na equação (3.4), é a forma mais comum de função de ativação utilizada na construção de RNAs.

$$f(x) = \frac{1}{1 + e^{-\lambda x}}, \tag{3.4}$$

onde $-\lambda x$ é a entrada a ser processada.

Haykin (2009) define a função sigmóide como uma função crescente, com o objetivo de realizar um balanceamento adequado entre comportamentos lineares e não-lineares em um intervalo entre [0,1], justificando sua grande usabilidade. No Código 1 é apresentada a implementação da função na linguagem de programação Python.

Haykin (2009) também evidência a utilização da tangente hiperbólica com o mesmo intuito da sigmóide, porém com um intervalo variando em um intervalo de [-1,1].

CÓDIGO 1 – Função sigmóide em Python

```
1 def sigmoid(self, entradas):
2     self.resultado = []
3     for entrada in entradas:
4         self.resultado.append(1/(1+math.exp(-entrada)))
5     return self.resultado
```

Resultados com esta característica são reconhecidos por formar um "S", exemplificado no Gráfico 1.

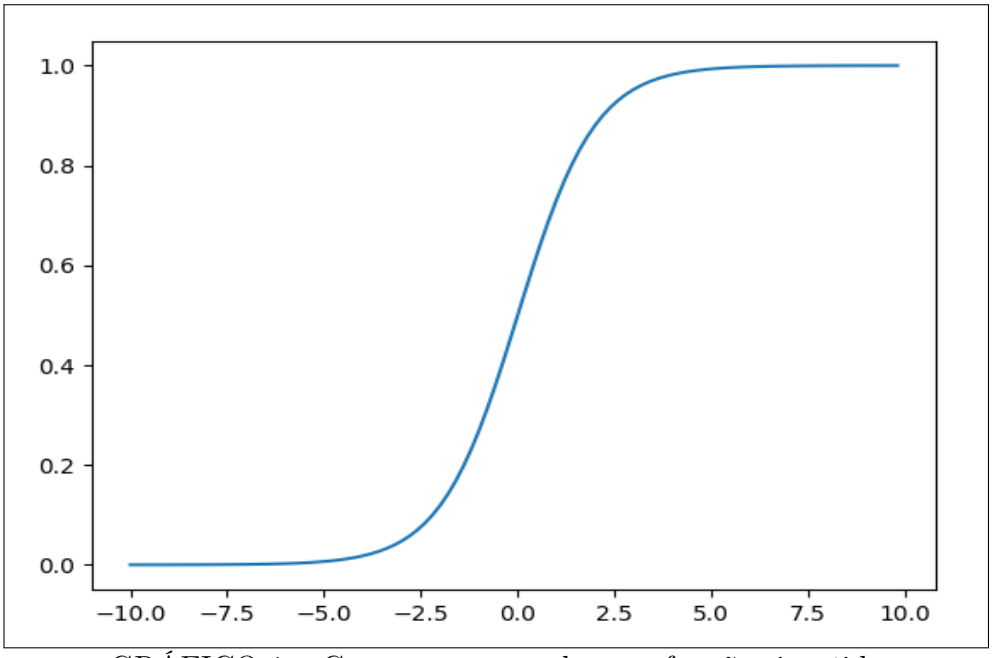


GRÁFICO 1 – Comportamento de uma função sigmóide
FONTE: Elaborado pelo autor

3.4 TREINAMENTO DE UMA REDE NEURAL ARTIFICIAL

Após determinar a arquitetura a ser utilizada pela RNA, a quantia de camadas, a função de ativação adequada ao trabalho a ser realizado, a rede neural deve adquirir conhecimento em relação aos dados aplicados sobre a mesma.

O treinamento de uma RNA consiste em minimizar uma função de custo, através de um algoritmo, cujos valores iniciais são escolhidos aleatoriamente com o objetivo de facilitar a busca pelo valor mínimo da função da amostra de treinamento, através de iterações.

O treinamento pode ser interrompido de duas formas: um limite de épocas estipulado para o treinamento ou um ponto de parada a partir de um critério de desempenho. Os critérios de desempenho típicos em um processo de treinamento, de uma determinada amostra de dados, são o Erro quadrático médio (EQM), Raiz do Erro Quadrático Médio (REQM) e o Erro Absoluto Médio (EAM) (GAMBOGI, 2013).

3.4.1 Processo de Aprendizagem

O processo de aprendizagem é considerado uma etapa de grande dificuldade em termos de análise de dados, porque a procura da solução adequada ocorre em um universo de soluções válidas de grande dimensão, onde a base de tomada de decisão serão os próprios dados treinados (MEDEIROS, 2004).

Para adquirir conhecimento, uma RNA requer um processo de inúmeras iterações, absorvendo os conhecimentos sobre o comportamento e relacionamento dos dados a cada ciclo e armazenando os ajustes realizados nos pesos sinápticos de cada neurônio (MENDONÇA NETO, 2014).

Segundo Haykin (2000), o processo de aprendizagem de uma rede neural pode ser resumido a uma sequência de três passos a cada estímulo enviado a rede, são eles:

1. *O* ambiente estimula a RNA com informações oriundas do mundo exterior;
2. *A* RNA experimenta alterações em seus pesos na tentativa de melhor responder ao estímulo recebido;
3. *A* RNA responde ao estímulo recebido após empreender as alterações realizadas no evento anterior.

A cada estímulo enviado a RNA em sua aprendizagem, a sequência acima é realizada, os pesos sinápticos são ajustados até que atinjam seus valores ideais, visando aumentar a acuracidade dos resultados do processo, este processo, segundo Haykin (2000), é conhecido como algoritmo de aprendizagem.

3.4.1.1 Aprendizagem Supervisionada

Nesse modelo de aprendizagem, o acompanhamento dos resultados é supervisionado constantemente durante o processo de treinamento e aprendizagem da rede. Nele, é disponibilizado o resultado esperado em relação aos dados aplicados na RNA, logo, estima-se a qualidade dos resultados obtidos de acordo com alguma métrica estabelecida, como, por exemplo, o desvio do padrão de resultados obtidos em relação aos aguardados (HAYKIN, 2009).

Então, um algoritmo de aprendizagem supervisionada busca usar explicitamente a informação atual para, no futuro, ser possível classificá-la como relevante ou não. O exemplo mais simples de um método baseado nesse tipo de aprendizagem é o modelo de regressão linear, quando estimado usando por exemplo a minimização dos erros quadráticos, isto é, a minimação do erro quadrático entre o valor da variável predita pelo modelo e o valor da variável real (ELPINIKI; KATARZYNA, 2016).

3.4.1.2 Aprendizagem não Supervisionada

No modelo de aprendizagem não supervisionada, não há supervisão dos dados processados, tal como não é disponibilizado um modelo de resultados esperados na aplicação em questão.

Segundo Haykin (2009), dentre as existentes destacam-se duas formas de aprendizagem não supervisionadas, são elas: aprendizagem por reforço e aprendizagem auto-organizada. A primeira delas está diretamente relacionada à métodos de programação dinâmica. Já a segunda, está relacionada a aprendizagem competitiva, método onde os neurônios são postos de forma que compitam entre si, sendo somente um deles ativado no ciclo de aprendizagem.

Portanto, esses métodos de aprendizagem não supervisionada podem ser utilizados em grandes quantidades de dados não estruturados para encontrar padrões existentes, e, a partir disso, realizar as devidas análises. Esta abordagem é bastante utilizada em aplicações de mineração de dados (*datamining*), onde o conteúdo de grandes bases de dados não é conhecido antecipadamente (ELPINIKI; KATARZYNA, 2016).

3.5 ALGORITMOS DE APRENDIZAGEM PARA TREINAMENTO DAS REDES

Muitos algoritmos são aplicados para a minimização e melhoria da aprendizagem da RNA. Esses algoritmos podem ser classificados como de minimização local e minimização global. Algoritmos de minimização local, tal qual o método gradiente descendente, são rápidos, porém convergem para mínimos locais. Por outro lado, os algoritmos de minimização global utilizam métodos de busca para achar soluções aproximadas, como

estratégias heurísticas e algoritmos genéticos, buscando não convergir para mínimos locais (HAYKIN, 2000).

3.5.1 *Backpropagation*

O algoritmo de retropropagação (*backpropagation*) é um método baseado no gradiente descendente, de aprendizado supervisionado e com alimentação à frente, que utiliza a função de ativação do tipo sigmóide e um coeficiente de aprendizado, denominado *learning rate*, responsável por especificar uma taxa de convergência da RNA, este coeficiente de aprendizado normalmente está em um intervalo entre [0,1] (HAYKIN, 2000). Neste modelo, reconhecido por ser a técnica mais utilizada dentre os métodos de treinamento aplicados, o erro de saída obtido se propaga para as camadas intermediárias da RNA, isso se dá pela necessidade de ajuste dos neurônios que não tem contato com a saída, possibilitando assim, a atualização dos pesos desses neurônios, aumentando o poder de classificação e de correção de erros (MEDEIROS, 2004).

O algoritmo usa a equação abaixo para atualizar os pesos de um determinado neurônio.

$$W_{ij,(t+1)} = W_{ij,t} + (\lambda)(\varepsilon W_{ij})(N_i) \tag{3.5}$$

A equação (3.5) realiza a atualização do peso W_{ij} de um determinado neurônio N_i referente ao neurônio atual N_j , onde o sub-índice t refere-se ao número de vezes em que a rede foi atualizada e λ refere-se à taxa de aprendizagem (*learning rate*). Esta taxa de aprendizagem é responsável por controlar o peso e suas alterações.

Como dito anteriormente, o algoritmo trabalha com a função de ativação sigmóide para tratar suas saídas, a equação (3.4) representa o cálculo realizado pela função.

Após organizar os dados da saída da rede, é necessário ajustar os pesos caso haja um erro na resposta esperada, este cálculo de erro é detalhado na equação (3.6)

$$E_j = \frac{1}{2} \sum_{saidas} (N_j - D_j)^2, \tag{3.6}$$

onde D_j é a saída desejada e N_j é o resultado atual do neurônio.

Encontrando o valor do erro, é necessário calcular a variação do mesmo em relação às mudanças dos valores de entrada, isso é feito determinando quanto a equação (3.6) se altera em relação à equação (3.5). O cálculo é dado pela equação abaixo.

$$\varepsilon S_j = E_j N_j (1 - N_j) \tag{3.7}$$

O próximo passo é calcular os pesos ajustados para W_{ij} da camada atual N_i para o neurônio atual N_j. Abaixo é possível observar o cálculo deste procedimento.

$$\varepsilon W_{ij} = \varepsilon S_j N_i \tag{3.8}$$

Por fim, a equação (3.9) é usada para calcular as atualizações dos pesos de cada neurônio N_j ajustado, dada por:

$$\varepsilon Hi = \sum_j \varepsilon S_j W_{ij}, \tag{3.9}$$

onde j representa os neurônios da camada escondida.

De forma geral, as operações de cada neurônio seguem os cálculos especificados acima. A ideia é que os valores dos neurônios sejam atualizados de acordo com cada entrada da rede.

3.5.2 Algoritmo de Aprendizado Hebbiano

O Aprendizado Hebbiano é uma classe de aprendizado não-supervisionado ou auto-organizado, onde uma RNA treinada sob este algoritmo possui capacidade de descobrir padrões significativos ou características em dados sem o auxílio de um padrão pré-definido, deixando a cargo da própria RNA aprender por sí só os padrões, características, correlações ou categorias nos dados de entrada (CASTRO, 1996).

Segundo Haykin (2000), o primeiro princípio do Aprendizado Hebbiano afirma que as modificações nos pesos sinápticos, baseadas em sinais pré-sinápticos e pós-sinápticos, tendem a se auto-amplificar. Uma sinapse forte leva à coincidência entre os sinais pré-sinápticos e pós-sinápticos e isto leva ao acréscimo da força da sinapse.

os sinais pré-sinápticos e pós-sinápticos, respectivamente x e y, podem ser equacionados por:

$$w_0(n+1) = w_0(n) + \eta y_0(n) x(n), \tag{3.10}$$

onde $w_0(n)$ é o peso atual do neurônio, η é a taxa de aprendizagem, $x(n)$ é a entrada da rede e $y_0(n)$ é a saída da rede.

Castro (1996) diz que o segundo princípio do Aprendizado Hebbiano trabalha através de competição entre os pesos sinápticos, ou seja, um aumento na força de algumas sinapses na rede deve ser compensado por um decréscimo em outras. Assim, só as sinapses de sucesso podem crescer, enquanto as de menos sucesso tendem a enfraquecer podendo, inclusive, desaparecer.

3.6 PROCESSO DE GENERALIZAÇÃO

O processo de generalização de uma RNA, ocorre a partir da sua capacidade em responder adequadamente não somente aos padrões de treinamento mas também aos demais padrões (MEDEIROS, 2004). Vale ressaltar que o processo de treinamento é uma das eta pas mais delicadas, em termos de análise e execução, ao se utilizar um modelo de RNA.

Tendo este conceito como ponto de partida, a capacidade de alta generalização desses modelos são fatores fundamentais para alcançar um nível de excelência. Desde os primórdios dessas técnicas, são utilizados métodos que possam abstrair a complexidade e suprir a necessidade de controlar o processo de generalização.

3.6.1 Problemas de Generalização

Diferentes realizações para um conjunto de treinamento podem fazer com que diferentes soluções sejam tomadas, dada uma topologia de rede. Essa variabilidade de soluções, dados os conjuntos de treinamentos distintos para uma mesma tarefa, é chamada de variância (HAYKIN, 2000).

Para garantir uma boa capacidade de generalização, a ideia é minimizar ao máximo essa variância que ocorre no processo de treinamento (MEDEIROS, 2004). As inconsistências que influenciam na capacidade de generalização de um algoritmo podem ser atribuídas aos seguintes fatores: sobre-ajuste (*overfitting*), sub-ajuste (*underfitting*) e sobretreinamento (*overtraining*). A seguir encontram-se alguns detalhes a respeito desses fatores.

3.6.1.1 *Overfitting* e *Underfitting*

As soluções que apresentam inconsistências devido à alta taxa de variância geralmente apresentam sobre-ajuste em relação aos dados de treinamento, efeito conhecido como overfitting (MEDEIROS, 2004). Na Figura 9 pode-se observar o comportamento de uma função com *overfitting*.

Uma forma de evitar a alta taxa de variância, e assim, a ocorrência de *overfitting*, se dá pela simplificação dos parâmetros de ativação da RNA. Porém, a redução desses parâmetros podem ocasionar outro problema muito comum dentre os protótipos desenvolvidos, denominado *underfitting*. *Underfitting* refere-se a um modelo que não pode nem modelar os dados de treinamento nem generalizar a novos dados, tornando-se um modelo estático, limitando assim, sua característica de aprender novos padrões (GEMAN; BIENESTOCK; DOURSAT, 1992). Na Figura 9 pode-se observar o comportamento de uma função com *underfitting*.

Com isso nota-se que deve existir um ponto de equilíbrio entre ambos efeitos, várias são as abordagens que tentam minimizar esses problemas, como os métodos construtivos e os algoritmos de *pruning*.

Os algoritmos construtivos buscam a construção gradual da RNA por meio da adição, até que um critério de parada envolvendo o erro de treinamento seja atingido. A resposta da rede se inicia em uma situação de *underfitting* e com a adição de novos neurônios aproxima-se de *overfitting*, a ideia desses algoritmos consiste em encontrar uma melhor generalização na transição entre esses dois extremos (MEDEIROS, 2004).

Os algoritmos de pruning, por sua vez, visam à minimização da estrutura pela eliminação gradativa dos pesos e dos neurônios, buscando diminuir os pesos excessivos, e assim, diminuindo o *overfitting*.

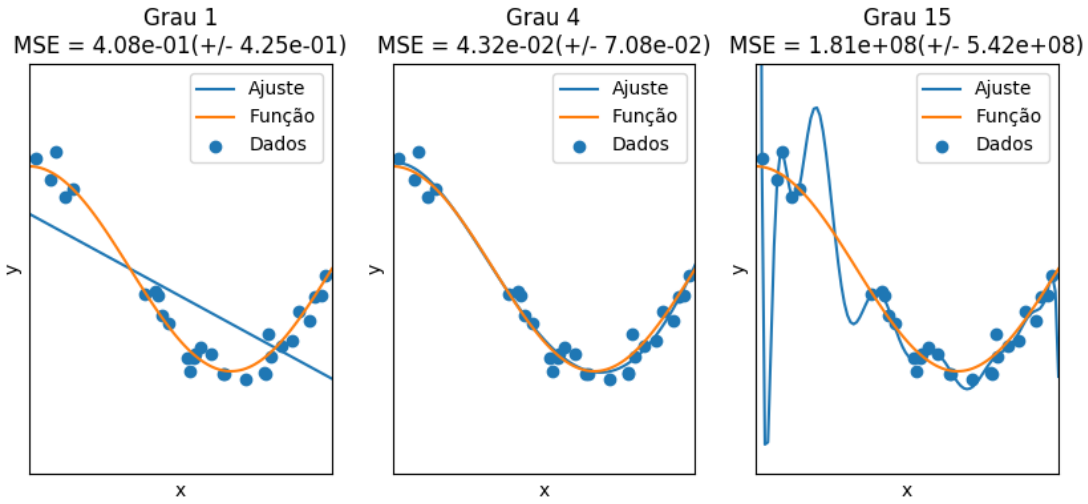


FIGURA 9 – Comportamento de funções com *overfitting* e *underfitting*
FONTE: Elaborado pelo autor

3.6.1.2 *Overtraining*

As arquiteturas convencionais de treinamento estão sujeitas a sofrerem um sobretreinamento (*overtraining*). Quando a rede parece estar respondendo o conjunto de dados cada vez melhor, ou seja, o erro de treinamento do conjunto de dados continua diminuindo, em algum ponto desse processo, a capacidade em resolver novos padrões torna-se falha. Isto ocorre por utilizar dados muito parecidos, assim como a quantidade de épocas em que esse conjunto de dados é treinado, viciando o conhecimento da RNA (HAYKIN, 2000).

3.7 TEORIA DA REGULARIZAÇÃO

A regularização é um método que busca melhorar a capacidade de generalização dos algoritmos de aprendizado por meio de alguma regra durante o processo de treinamento. A ideia básica da regularização é estabilizar a solução por meio de uma suavização entre o mapeamento da entrada até a saída, no sentido de que entradas similares correspondam a saídas similares. A suavização pode transformar um problema mal posto em um problema bem posto (MEDEIROS, 2004). Um dos problemas a ser resolvido com métodos de regularização, seria encontrar um ponto de equilíbrio entre *overfitting* e *underfitting* no processo de treinamento de uma RNA, implicando na suavização do conjunto de dados.

3.8 SÉRIES TEMPORAIS

Séries temporais são definidas como sequências de dados indexados ao longo de um período. Diferentemente de dados em seção transversal (*cross-section*), em que os dados são coletados somente em um determinado instante de tempo, representando uma instância dos dados naquele momento, a ordem temporal dos dados tem relevância na obtenção de resultados e conclusões. Se a ordem dos dados for modificada, haverá perda de informação relevante sobre os dados, podendo obter conclusões equivocadas em relação à série em estudo (MENDONÇA NETO, 2014).

As séries temporais podem ser classificadas de diversas formas. Elas podem ser contínuas, quando a sequência de dados é descrita continuamente no tempo, e discretas, quando a sequência de dados é descrita em instantes de tempo discretos, havendo um espaçamento entre os instantes. A classificação em discreta e contínua é em relação ao tempo e não ao valor de cada observação. Pode-se ter uma série temporal discreta, mas o conteúdo das suas variáveis ser contínuo. Na prática, a utilização de séries temporais econômicas e financeiras é dada na forma discreta, pois, a coleta de dados é viabilizada em instantes de tempo discretos (OLIVEIRA, 2012).

3.9 MERCADO DE CAPITAIS

O mercado de capitais é um sistema de distribuição de valores mobiliários que visa proporcionar liquidez aos títulos de emissão de empresas e viabilizar seu processo de capitalização. É constituído pelas bolsas, corretoras e outras instituições financeiras autorizadas (TORORADAR, 2015).

Os mercados de grandes ações são considerados os pilares centrais do crescimento e desenvolvimento econômico. Prever como irá se comportar futuramente o movimento de ações, tornou-se interesse comum entre investidores deste mercado (PEREIRA, 2014).

Partindo deste pretexto, estudos ligados a predição de valores do mercado acionário, dividem-se basicamente em três vertentes. Há os que não crêem que investidores podem conseguir consideráveis vantagens em transações, que se fundamentam em teorias aleatórias de mercado. Há também os que crêem no hábito de seguir aspectos relacionados a análise fundamentalista de empresas e indicadores macro-econômicos. Por fim, há os que crêem que com base em séries recentes de dados e históricos é possível conseguir generalizar e classificar padrões de dados. A partir da última citação, surgem os interesses em trabalhos que relacionam o tema à técnicas de inteligência computacional, especificamente o uso de redes neurais artificiais (PEREIRA, 2014).

3.9.1 Análise Fundamentalista de Acões

Antes da realização de qualquer investimento no mercado acionário é fundamental que sejam analisados itens de considerável importância no ramo financeiro, como indicadores de balanço e de mercado. O objetivo da análise é de reconhecer previamente as altas e baixas de determinadas ações em um período de tempo.

A análise fundamentalista busca, basicamente, avaliar a saúde financeira das empresas, projetar seus resultados futuros e determinar o preço justo para as suas ações. Para isso, os analistas levam em consideração os chamados fundamentos da empresa, isto é, todos os fatores macro e microeconômicos que influenciam no seu desempenho. A partir de uma minuciosa análise de todos eles, é possível projetar os resultados da companhia a longo prazo, em geral num período de cinco a dez anos (EXAME, 2010).

3.9.2 Análise Técnica de Ações

Análise Técnica de ações é a prática de medir as flutuações futuras do preço de uma ação analisando suas atividades passadas. Este método envolve a procura de padrões gráficos e o exame de outros dados históricos relacionados a preço e volume de ações negociadas (TORORADAR, 2015).

Relaciona as oscilações de preços do mercado em relação à ação, e não à empresa. Este modelo de análise acredita na repetitividade do comportamento humano e no poder da ciência estatística como forma de determinar, com base no comportamento passado, as perspectivas para o mercado no futuro (PEREIRA, 2014).

A análise técnica também é conhecida como escola grafista, e auxilia o investidor na escolha do melhor momento para compra e venda de ações. Esta escola baseia-se na análise gráfica, tendo como base os volumes e os preços pelos quais foram comercializadas as ações nos pregões anteriores (FORTUNA, 2008).

Pensando dento deste contexto, não seria extremamente lucrativo saber qual lado está mais forte? O da demanda (compradores) ou da oferta (vendedores)? Esse é o trabalho da Análise Técnica de Ações.

Uma boa métrica para análise técnica se dá pela utilização de seus gráficos de preço das ações para encontrar padrões que indiquem quem está mais forte no mercado. Se existem sinais de que a demanda está forte e a oferta fraca pode ser uma bela oportunidade de comprar e ganhar com a alta, enquanto no cenário contrário pode ser que seja a hora ideal de se vender aquela ação (TORORADAR, 2015).

3.9.3 Bolsa de Valores NASDAQ

A bolsa de valores que será utilizada como referência para a execução do trabalho, será a *National Association of Securities Dealers Automated Quotations* (NASDAQ).

Fundada em 1971, é atualmente a segunda maior bolsa de valores de mercado do mundo. Grande parte das empresas listadas no mercado de ações da NASDAQ, fazem parte do ramo de produção de alta tecnologia, como o Facebook, Apple, Amazon, Adobe, Cisco, dentre outras, incluindo a própria NASDAQ, listada em seu mercado desde 2002. A NASDAQ é muito conhecida pela realização de negócios de mercado inteiramente de forma eletrônica. O mercado financeiro considera o conjunto de negócios realizados através da NASDAQ como Nova Economia, o que a difere da Bolsa de Valores de Nova York, por exemplo, considerada como Velha Economia (CHRISTIE; SCHULTZ, 2003).

4 MATERIAIS E MÉTODOS

Conforme mencionado anteriormente, o objetivo principal dessa dissertação é especificar e aplicar um modelo de RNA para análise e predição de valores acionários. Neste capítulo será realizada a pergunta de pesquisa e dos objetivos do trabalho na escolha da metodologia. Em seguida, esta metodologia será descrita e classificada quanto ao seu conteúdo e quanto aos métodos empregados na coleta e análise de dados.

4.1 IDENTIFICAÇÃO DA PESQUISA

Para se escolher a metodologia de pesquisa, alguns aspectos importantes precisaram ser analisados. Uma pesquisa pode ser realizada com dados criados ou com dados existentes. No primeiro caso, os dados são coletados após uma intervenção destinada a provocar uma mudança. No segundo, os dados estão presentes na situação em estudo e o pesquisador, por meio das técnicas de pesquisa, faz tais dados aparecerem, sem a intenção de modificá-los, através de uma intervenção. Na pesquisa experimental, delimita-se o fenômeno, formulam-se hipóteses, determinam-se os métodos e submete-se o fenômeno à experimentação em condições de controle (LAVILLE; DIONNE, 1999).

A escolha de uma estratégia de pesquisa tem que ser feita considerando-se, entre outros elementos, a natureza da questão da pesquisa, o contexto no qual a pesquisa se realizará, a formação e a experiência do pesquisador. De maneira geral, pode-se dividir as pesquisas em quantitativas ou experimentais e qualitativas (LAVILLE; DIONNE, 1999).

Portanto, de acordo com as afirmativas acima, a presente pesquisa é caracterizada por ser experimental e qualitativa. Segundo Chizzotti (1991), na pesquisa experimental, o pesquisador parte de um estado delimitado a priori, sobre qual cria possíveis hipóteses que podem ser analisadas, determinando os métodos de verificação a serem utilizados, através dos quais procurará controlar as condições do experimento.

4.2 SELEÇÃO DAS AÇÕES

Para iniciar o processo de desenvolvimento deste trabalho, é necessário definir quais ações serão escolhidas para compor a implementação. Os critérios para definir quais serão selecionadas, são caracterizados pelo alto potencial de movimentação e lucratividade na NASDAQ, ou seja, são as empresas que mais influênciam na ecônomia, que, por coincidência, são do ramo da tecnologia (CHRISTIE; SCHULTZ, 2003).

Tendo em vista os critérios especificados acima, as empresas selecionadas e o código de suas ações para a realização da coleta dos dados são:

- Apple (AAPL);
- Amazon (AMZN);
- Cisco Systems (CSCO);
- Intel (INTC);
- Microsoft (MSFT);

4.3 TÉCNICA PARA COLETA DE DADOS

Com a definição da bolsa de valores em questão e das empresas que serão utilizadas para realizar a implementação deste trabalho, a próxima etapa se dá pela necessidade em coletar seus dados histórios.

Como os dados históricos das ações caracterizam-se por serem séries temporais, o tipo de técnica aplicada para conseguir indicadores de relevância para uma boa análise dos objetivos propostos, refere-se à análise técnica. Portanto, esta fase de coleta de dados é muito importante para o bom funcionamento da RNA que será criada, tornando-se essencial para o êxito final do trabalho.

4.3.1 Definição das Séries

Selecionar os dados corretos para trabalhar com RNAS é uma etapa de grande relevância, pois são esses dados que serão treinados pela mesma e que irão garantir a sua estabilidade e capacidade de generalização.

Sendo assim, a partir do conhecimento concebido pelo Capítulo 3, pode-se destacar dois aspectos importantes para a análise de quais dados coletar dentro de um problema em geral, sendo eles:

1. Dados que influeciam diretamente no resultado final e na coêrencia da análise proposta;
2. Uso de múltiplas séries de um mesmo dado para que a rede não se sujeite à *overfitting* ou *underfitting*.

Segundo Tororadar (2015), as variáveis que influenciam em um sistema de análise técnica das ações são:

1. Abertura: Valor da ação no ínicio do dia;
2. Máximo: Valor máximo negociado no dia;
3. Mínimo: Valor mínimo negociado no dia;

4. Volume: Número de negociações do papel no dia (valor bruto das negociações);
5. Valor de fechamento no dia;
6. Médias móveis simples: As médias móveis suavizam os dados de preços para formar um indicador de tendência sequencial. Elas não prevêem a direção dos preços, mas, antes, definem a sua direção atual com um atraso. Como o próprio nome indica, uma média móvel é uma média que se move. Os dados antigos são retirados, a medida que dados mais recentes se tornam disponíveis, isto faz com que a média se mova ao longo do tempo. A maioria das médias móveis são baseadas em preços de fechamento.
7. *Moving Average Convergence Divergence* (MACD): Indicador que controla a divergência e convergência das médias móveis. O MACD é formado pela diferença entre duas médias móveis, sendo uma de longo prazo e outra de curto prazo. Também, segundo Tororadar (2015), as médias móveis utilizadas para o cálculo do MACD não possuem restrições, desde que, os valores de curto e longo prazo sejam respeitados. Portando, para o presente trabalho, será calculado o MACD referente às médias de 10 e 26 dias, de longo e curto prazo, respectivamente.

4.4 FERRAMENTAS PARA COLETA DE DADOS

Com a especificação das séries que serão coletadas, é necessário preparar um ambiente que dispõe o acesso à esses dados de forma simplificada.

Pensando nesse contexto, é essencial utilizar uma linguagem de programação que tenha suporte à esse modelo de busca por dados. Tendo em vista esta premissa, a disponibilidade de uma API ou biblioteca é de extrema importância para a automatização deste processo.

4.4.1 Linguagem de Programação Python

Segundo McKinney (2013), apesar das diversas linguagens de programação disponíveis hoje para o desenvolvimento de aplicações e *scripts*, a linguagem de programação Python vem se destacando e crescendo constantemente na comunidade científica, seja por sua variedade de bibliotecas que auxiliam os desenvolvedores e pesquisadores à resolverem seus problemas de forma mais simplificada, até seu poder computacional e suas características de multiplataforma e multiparadigma.

Um fator que determina a utilização da linguagem de programação Python, como dito no paragrafo anterior, se dá pela sua variabilidade de bibliotecas voltadas, principalmente, para a análise de dados. Colocando isso no contexto deste trabalho, é essencial o uso de uma ferramenta que proporcione um alto nível de funcionalidades disponíveis para serem exploradas.

4.4.2 Biblioteca Pandas

A biblioteca pandas é um pacote disponível na linguagem de programação Python que fornece fácil acesso à dados disponíveis na internet. O objetivo principal do pandas é, justamente, fornecer buscas flexíveis e intuitivas para a análise de dados estruturados e não estruturados. Além disto, busca como objetivo específico, se tornar a ferramenta de análise, busca e manipulação de dados, aberta, mais poderosa e flexível existente (PANDAS, 2017).

Segundo Pandas (2017), o seu uso é adequado para diferentes tipos de dados, sendo eles:

1. Dados tabulares com colunas, como em uma tabela *Structured Query Language* (SQL) de um banco de dados relacional ou uma planilha do Excel;
2. Dados de séries temporais não ordenados;
3. Qualquer outra forma de conjuntos de dados observacionais e estatísticos.

Portanto, a principal justificativa para a utilização da biblioteca se dá pelo seu desenvolvimento, de código aberto, que impulsiona a comunidade a realizar diferentes aplicações, além da sua excelente documentação e suporte de acesso aos dados históricos da bolsa de valores NASDAQ, que são requisitos para a realização deste trabalho.

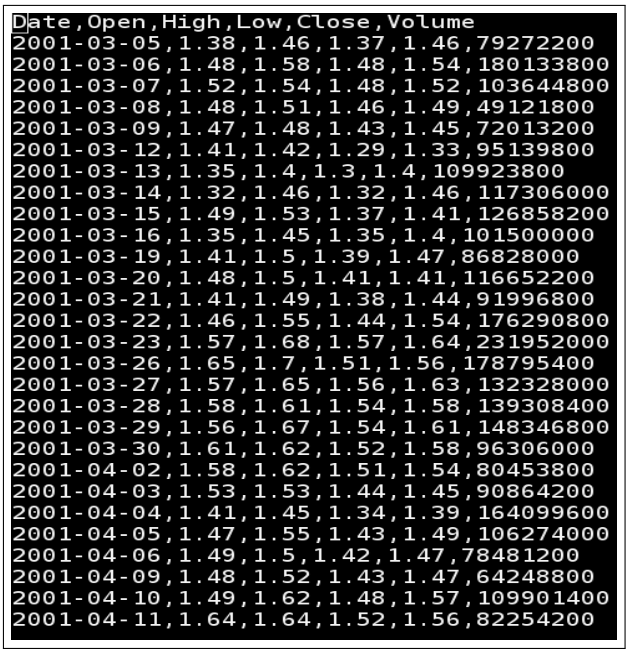
Para exemplificar como a biblioteca facilita o processo de coleta dos dados, o Código 2 ilustra uma pesquisa das ações da Microsoft, no período de 05/03/2001 a 11/04/2001.

CÓDIGO 2 – *Script* para coleta de dados

```
1 import pandas_datareader.data as web
2 import datetime
3
4 start = datetime.datetime(2001, 03, 05)
5 end = datetime.datetime(2001, 04, 11)
6
7 dataset = web.DataReader("MSFT", 'google', start, end)
8 dataset.to_csv('~/Documentos/TCC/dist-tcc/Implementacao/dados/msft.csv')
```

As séries históricas são disponibilizadas em arquivos com valores separados por vírgulas (*.csv), contendo a data de negociação, valor de abertura, valores máximo e mínimo atingidos, valor de fechamento e o volume de negociações. Este formato de de arquivo, segundo Pandas (2017), é chamado de *DataFrame*.

Na Figura 10 é possível observar o arquivo criado a partir do Código 2, demonstrando o formato que foi especificado no código e o que a API retorna para a manipulação.



```
Date,Open,High,Low,Close,Volume
2001-03-05,1.38,1.46,1.37,1.46,79272200
2001-03-06,1.48,1.58,1.48,1.54,180133800
2001-03-07,1.52,1.54,1.48,1.52,103644800
2001-03-08,1.48,1.51,1.46,1.49,49121800
2001-03-09,1.47,1.48,1.43,1.45,72013200
2001-03-12,1.41,1.42,1.29,1.33,95139800
2001-03-13,1.35,1.4,1.3,1.4,109923800
2001-03-14,1.32,1.46,1.32,1.46,117306000
2001-03-15,1.49,1.53,1.37,1.41,126858200
2001-03-16,1.35,1.45,1.35,1.4,101500000
2001-03-19,1.41,1.5,1.39,1.47,86828000
2001-03-20,1.48,1.5,1.41,1.41,116652200
2001-03-21,1.41,1.49,1.38,1.44,91996800
2001-03-22,1.46,1.55,1.44,1.54,176290800
2001-03-23,1.57,1.68,1.57,1.64,231952000
2001-03-26,1.65,1.7,1.51,1.56,178795400
2001-03-27,1.57,1.65,1.56,1.63,132328000
2001-03-28,1.58,1.61,1.54,1.58,139308400
2001-03-29,1.56,1.67,1.54,1.61,148346800
2001-03-30,1.61,1.62,1.52,1.58,96306000
2001-04-02,1.58,1.62,1.51,1.54,80453800
2001-04-03,1.53,1.53,1.44,1.45,90864200
2001-04-04,1.41,1.45,1.34,1.39,164099600
2001-04-05,1.47,1.55,1.43,1.49,106274000
2001-04-06,1.49,1.5,1.42,1.47,78481200
2001-04-09,1.48,1.52,1.43,1.47,64248800
2001-04-10,1.49,1.62,1.48,1.57,109901400
2001-04-11,1.64,1.64,1.52,1.56,82254200
```

FIGURA 10 – Representação dos dados coletados pela biblioteca pandas
FONTE: Elaborado pelo autor

4.4.3 Google *Finance* API

É importante citar qual a fonte onde o pandas busca as séries históricas, por questões como confiabilidade dos dados. O acesso é realizado através da integração com o Google *Finance* API, um banco de dados quase em tempo real mantido pelo Google, que trabalha com as principais bolsas de valores do mundo (PANDAS, 2017).

A API trabalha com um limite inicial para a coleta das ações, a partir de 03/05/2001 e vai se atualizando diariamente. Tendo em vista que para trabalhar com RNAs é importante buscar o máximo de informações possíveis e, também, considerando o limite disponível pela API, o período das ações das empresas selecionadas serão coletadas iniciando no ano de 2001 até o ano de 2017.

Sendo assim, a ideia é preparar um ambiente que proporcione o uso dessa ferramenta, junto à biblioteca citada na Subseção 4.4.2 para realizar as buscas pelos dados necessários.

4.4.4 PyCharm IDE

É necessário utilizar uma *Integrated Development Environment* (IDE) para auxiliar no *script* de coleta de dados. PyCharm é uma IDE proprietária do grupo JetBrains que conta com uma versão livre para estudantes, que será realizada neste trabalho.

PyCharm suporta o desenvolvimento com a linguagem de programação Python desde a sua versão 2.4 até a versão 3.6. Vale citar que outras IDEs para o desenvolvimento poderiam ser utilizadas, porém a praticidade e a boa documentação das ferramentas JetBrains são fatores que justificam a sua utilização (PYCHARM, 2017).

4.5 ESPECIFICAÇÃO DO MODELO DE REDES NEURAIS ARTIFICIAIS

Tendo como base o conhecimento adquirido no Capítulo 3, a definição de um modelo de RNA varia muito, sendo específico para cada problema do mundo real. Por isso, é importante frisar que não existe um modelo pré-definido e eficiente em RNAs, ficando a cargo da análise dos resultados, e, muitas vezes, por métodos de tentativa e erro para se chegar à um resultado eficiente (HAYKIN, 2000).

Porém, com o estudo realizado no Capítulo 3, esse processo pode ser menos custoso do que parece. Avaliar o problema em que se está trabalhando, junto com o conhecimento teórico, é de grande auxílio para a especificação e construção de um modelo inicial que se aproxime do desejado.

Para especificar quais modelos não devem ser aplicados, ou seja, modelos que não tenham características referente a resolução do problema em questão, algumas perguntas-chave serão elaboradas, sendo elas:

1. "Qual o modelo de dados se está trabalhando?";
2. "Estes dados serão treinados a partir de resultados esperados?";
3. "A RNA será exposta à uma quantidade significativa de dados desconhecidos aos treinados?".

A primeira pergunta refere-se ao modelo de transmissão dos dados que a RNA deve operar, ou seja, se a mesma será formada por camadas adiante ou recorrente. Como no presente trabalho serão analisados dados através de séries temporais, segundo Haykin (2000), a RNA deverá trabalhar com um modelo de camadas alimentada adiante (*Feedforward*).

A segunda pergunta refere-se ao processo de aprendizagem que deve ser utilizado. Se os dados são treinados a partir de resultados esperados, a RNA deverá possuir um método de aprendizado supervisionado, caso contrário, um modelo de RNA com treinamento não supervisionado deverá ser utilizado. Para este trabalho, a RNA será treinada a partir dos resultados esperados através das séries temporais, ou seja, o modelo trabalhará com aprendizagem supervisionada.

A terceira pergunta refere-se à capacidade de generalização. Sendo assim, um modelo de RNA que irá trabalhar com dados desconhecidos deve ter caracteristícas de boa generalização, caso contrário, um modelo simples já é o suficiente.

4.5.1 Definição da Arquitetura

Tendo em vista os critérios estabelecidos na seção anterior, chega-se a definição de que a RNA deverá possuir uma arquitetura de camadas alimentada adiante, um mé-

todo de aprendizado supervisionado e uma função de custo, para controlar o processo de generalização da mesma.

Haykin (2000) direciona este modelo de rede ao *Perceptron* de Múltiplas Camadas (MLP, *Multilayer Perceptron*), são redes neurais compostas por um conjunto de unidades sensoriais que constituem a camada de entrada, uma ou mais camadas ocultas e uma camada de saída, onde o sinal de entrada se propaga para frente através da rede, camada por camada. Redes MLP trabalham normalmente junto ao algoritmo *backpropagation* para realizar seu treinamento e aprendizado.

Tendo em vista a especificação do modelo da rede e dos dados que serão utilizados para realizar o treinamento e os testes, a próxima etapa será definir quais serão as entradas utilizadas. Como o objetivo do trabalho é realizar a predição do valor de abertura das ações em dias posteriores aos treinados, todas as váriaveis coletadas serão utilizadas como entrada da RNA. Portanto, os parâmetros que irão compor a camada de entrada da RNA são:

1. Valor de abertura;
2. Valor máximo negociado no dia;
3. Valor mínimo negociado no dia;
4. Valor de fechamento;
5. Número de negociações do papel;
6. Média móvel simples de 10 dias;
7. Média móvel simples de 26 dias;
8. *Moving Average Convergence Divergence* (MACD).

Segundo Haykin (2000) e Marangoni (2010), na camada intermediária da RNA existe um cuidado especial para especificar sua composição, pois o número de neurônios depende de várias condições, tais como:

- Quantidade de dados para treinamento em relação a quantidade de neurônios na camada oculta: Deve-se ter cuidado para não utilizar neurônios e camadas demais, o que pode levar a rede a memorizar os padrões de treinamento, quebrando seu processo de aprendizagem (*overfitting*). O inverso também é válido, poucos neurônios para uma grande quantidade de dados, pode levar a rede a não encontrar uma solução desejável, pelo pouco poder de processamento (*underfitting*);

- Definir o número de neurônios em função da dimensão das camadas de entrada e saída da rede: Pode-se definir o número de neurônios na camada escondida como sendo a média aritmética entre o tamanho da entrada e da saída da rede;
- Utilizar um número de sinapses dez vezes menor que o número de exemplos disponíveis para treinamento.

Por isso, para a grande maioria dos problemas utiliza-se apenas uma camada escondida, onde sua quantidade de neurônios é definida buscando um meio termo entre a quantidade de entradas e a sua saída, tentando evitar as complicações evidenciadas nos tópicos anteriores.

Mediante estas informações, o modelo será desenvolvido com apenas uma camada intermediária composta por 13 neurônios, buscando otimizar a sua estrutura e obtendo um resultado eficiente para o problema.

Por fim, a saída da rede será composta por uma única camada que irá conter o valor de abertura do dia seguinte. É importante deixar claro que este valor, durante o processo de treinamento, será preenchido com os valores reais das ações em relação ao período que será escolhido para o processo de treinamento.

Na Figura 11 é possível visualizar a composição gráfica da arquitetura proposta.

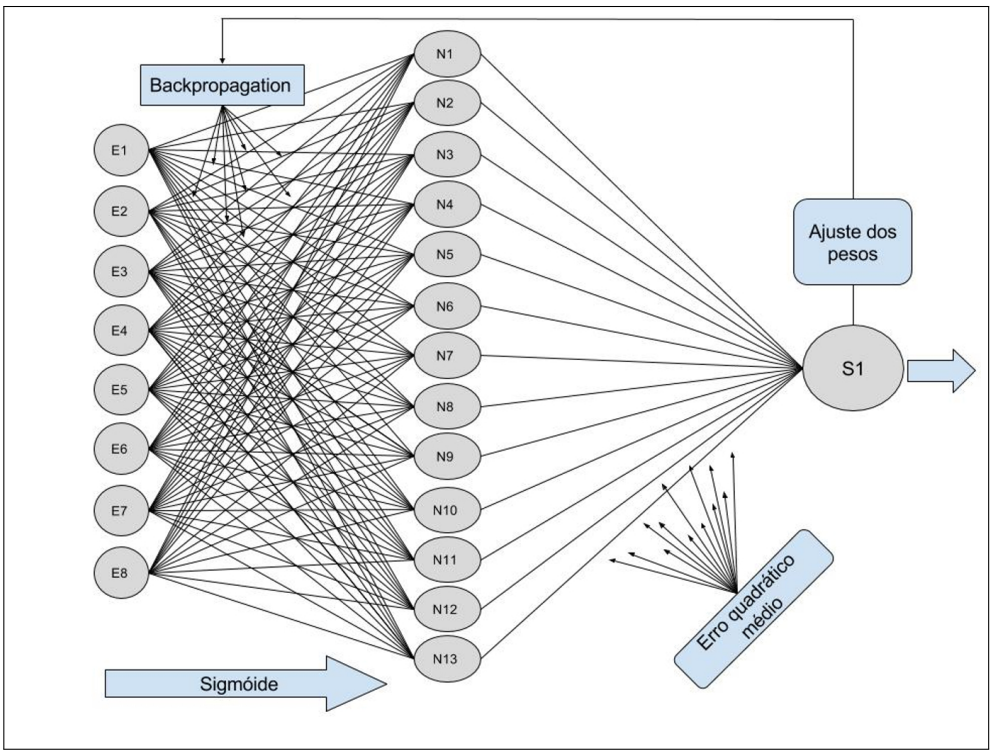


FIGURA 11 – Modelo de arquitetura proposta utilizando o algoritmo *backpropagation*
FONTE: Elaborado pelo autor

4.6 DEFINIÇÃO DA BASE DE TREINAMENTO

A partir do modelo de rede definido, é necessário classificar os dados que serão utilizados na fase de treinamento e os que serão utilizados para a realização dos testes. Como no trabalho serão coletadas as ações de 5 empresas distintas, a ideia é que para cada empresa, individualmente, sejam realizados os processos de treinamento e testes, com objetivo de observar o comportamento da rede nas mais diversas situações e proporcionar uma análise mais detalhada do modelo.

Os dados utilizados para o treinamento da rede devem ser significativos e cobrir amplamente o domínio do problema, e não apenas envolver os casos de operações normais ou rotineiras, mas também incorporar casos de exceção e de situações limites. Seguindo o funcionamento do algoritmo *backpropagation*, os dados treinados devem seguir os seguintes passos:

1. Aplicar o vetor de entrada da rede;
2. Calcular a saída da rede;
3. Ajustar os pesos da rede de maneira a minimizar o erro;
4. Repetir o passo (1) até o passo (3) para cada vetor do conjunto de treinamento, até que o erro (função de custo) seja satisfeito.

Segundo Haykin (2000), ao determinar a classificação de um conjunto para treinamento, devem ser levados em consideração os seguintes fatores:

- Quando há uma grande quantidade de amostras disponíveis para o treinamento, a divisão deve se aproximar de 33% para treino e 67% para testes;
- Quando o número de amostras é reduzido deve-se diminuir a proporção do conjunto de testes, com um valor próximo de 60% para treinamento e 40% para testes;
- Em casos com um número muito pequeno de amostras, é recomandável utilizar 50% para ambos.

4.7 BIBLIOTECA PYBRAIN

Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library (PyBrain) é uma biblioteca de aprendizagem de máquinas para Python. Seu objetivo é oferecer algoritmos flexíveis e fáceis de usar que proporcione uma forma mais intuitiva de desenvolver algoritmos complexos relacionados à Inteligência Artificial. Embora existam poucas bibliotecas de aprendizado de máquinas, PyBrain pretende ser uma

biblioteca modular fácil de aprender, podendo ser utilizada por estudantes de nível básico até um nível mais avançado, oferecendo flexibilidade e algoritmos para pesquisa de ponta (SCHAUL et al., 2010).

Segundo Schaul et al. (2010), como o próprio nome sugere, PyBrain contém algoritmos para redes neurais, aprendizagem supervisionada, aprendizagem não supervisionada e algoritmos evolutivos. A biblioteca é construída com seu núcleo baseado em redes neurais, onde todos os métodos de treinamento aceitam uma rede como instância pré-treinada. Isso faz de PyBrain uma ferramenta poderosa para a execução de tarefas complexas de serem resolvidas.

A Figura 12 demonstra, de maneira mais clara, como a biblioteca PyBrain irá auxiliar no projeto.

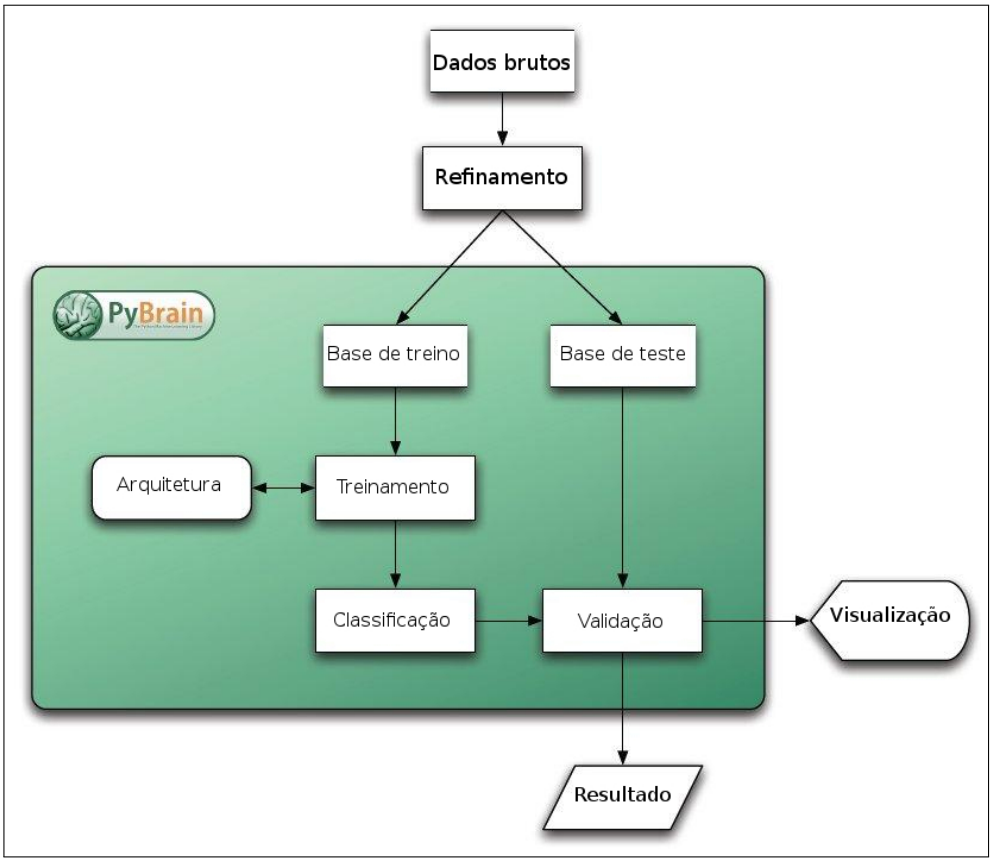


FIGURA 12 – Arquitetura da biblioteca PyBrain aplicada na estrutura do trabalho
FONTE: Adaptado de Schaul et al. (2010)

PyBrain tem ênfase nas arquiteturas de rede, que podem ser treinadas e manipuladas com quase todos os algoritmos oferecidos de aprendizado de máquina. As redes são compostas por módulos que estão ligados através de conexões, onde os nós são os módulos e as bordas são as conexões. Isso torna o PyBrain muito flexível, mas também necessário em todos os casos, seguir este padrão.

Alguns exemplos de arquiteturas disponibilizada pela biblioteca são:

- Redes Neurais Alimentadas Adiante (*Feedforward*);
- Redes Neurais Recorrentes;
- Redes Neurais Recorrentes Multidimensionais;
- Redes de Kahonen (Mapas Auto-Organizáveis);
- Redes Bidirecionais;
- Topologias Personalizadas.

Dentre os algoritmos de aprendizagem supervisionada utilizado pelo PyBrain, destacam-se:

- Algoritmo de retropropagação (*backpropagation*);
- Algoritmo de R-prop;
- Algoritmos de Máquinas de Vetores de Suporte (SVM).

Todos os algoritmos e implementações matemáticas da PyBrain tem como base o SciPy. SciPy é um ecossistema baseado em Python, de código aberto, para matemática, ciência e engenharia. Sua biblioteca central é o NumPy, que fornece uma manipulação conveniente e rápida de estruturas de dados que armazenam os valores em mais de uma dimensão. Portanto, a biblioteca SciPy foi desenvolvida para trabalhar com *arrays* NumPy, fornecendo rotinas amigáveis e eficientes para realizar cálculos complexos (JONES et al., 2001).

Também é importante evidenciar que a biblioteca PyBrain é de código aberto e gratuita para todos, licenciada sob a *Berkeley Software Distribution* (BSD) Software.

4.8 BIBLIOTECA MATPLOTLIB

O matplotlib é uma biblioteca disponível para a linguagem de programação Python que contém recursos para a geração de gráficos 2D, a partir de um determinado conjunto de dados. Seus gráficos são criados com alta qualidade utilizando comandos simples e intuitivos (HUNTER, 2007).

Segundo Hunter (2007), o código Matplotlib é dividido conceitualmente em três partes:

- pylab: conjunto de funções disponíveis em matplotlib.pylab que permite a escrita de códigos simples de serem criados;

- API: camada responsável por realizar toda a lógica da biblioteca, criando por exemplo figuras, textos, gráficos e linhas;
- *backend*: conjunto de funções que dependem do dispositivo de saída, como por exemplo imagens e arquivos no formato *Portable Document Format* (PDF).

Desta forma, todos os métodos necessários para realizar a implementação deste trabalho foram levantados, o próximo capítulo irá detalhar o processo de desenvolvimento das técnicas, e, consequentemente, da obtenção dos objetivos desejados, levando todos os aspectos estudados e definidos até o momento considerado.

5 IMPLEMENTAÇÃO DAS TÉCNICAS

Este capítulo tem como objetivo apresentar, de forma mais detalhada, todo o processo de implementação realizado no presente trabalho, com o intuito de alcançar os objetivos especificados no Capítulo 1.

5.1 REALIZAÇÃO DA COLETA DE DADOS

Tendo em vista as ferramentas necessárias para realizar à coleta das séries temporais que serão utilizadas para o treinamento e testes da RNA, foi desenvolvido um método de automatização de aquisição dos *DataFrames* das ações que serão coletadas para a análise.

Inicialmente, foi criada uma interface em Python que contém a assinatura do método que as classes das empresas deverão implementar. Após isso, foram criadas as classes para as respectivas empresas, definidas no Seção 4.2. Para exemplificar o que está sendo exposto, o Código 3 demonstra, de forma mais intuituva, a implementação do atual procedimento.

CÓDIGO 3 – Implementação da interface Empresa em Python

```
1  import abc
2  from seriestemporais.stack_empresas.crawler import *
3
4  class Empresa(metaclass=abc.ABCMeta):
5
6      @abc.abstractmethod
7      def executa_busca(self):
8          pass
9
10 class Apple(Empresa):
11
12     def __init__(self):
13         self.nome_empresa = 'apple'
14         self.codigo = 'AAPL'
15         self.executa_busca()
16
17     def executa_busca(self):
18         crawler = Crawler(self.nome_empresa, self.codigo)
19         crawler.executa_busca()
```

Analisando este código, pode-se observar o desenvolvimento da interface "Empresa" e da classe "Apple", que implementa esta interface através do parâmetro (Empresa) em sua definição. Também é possível observar, na classe Apple, a implementação

do método construtor "__init__" que contém seu respectivo nome e código, necessário para realizar a busca. O método "executa_busca", por sua vez, faz uma chamada ao objeto que realizará a coleta das respectivas ações, passando seus parâmetros como valor. O Código 4 ilustra a implementação do objeto de busca.

CÓDIGO 4 – Implementação do objeto Crawler que realiza a busca das ações

```
"""
Crawler.py: Buscador genérico das séries temporais.
"""
__author__ = 'Fernando Demarchi Natividade Luiz'
__email__ = "nativanando@gmail.com"
__version__ = "0.0.1"

import pandas_datareader.data as web
import datetime

class Crawler:

    def __init__(self, nome_empresa, codigo):
        self.start = datetime.datetime(2001, 1, 1)
        self.end = datetime.datetime(2017, 8, 31)
        self.nome_empresa = nome_empresa
        self.codigo = codigo

    def executa_busca(self):
        file = web.DataReader(self.codigo, 'google', self.start, self.end)
        file.to_csv('~/Documentos/TCC/dist-tcc/Implementacao/dados/' + self.nome_empresa + '.csv')
```

O Código 4 utiliza a biblioteca pandas para realizar a coleta das ações. Pode-se observar que o método construtor "__init__" recebe o nome e o código da empresa que faz a chamada de sua instância. Também é possível analisar o método "executa_busca" implementado, que realiza a chamada da função "web.DataReader" passando quatro parâmetros, sendo eles:

1. Código da empresa;
2. API que será realizada a busca;
3. Data de inicial da coleta;
4. Data final da coleta.

Após a realização da requisição, a API retorna um arquivo com as ações entre as respectivas datas, no formato especificado na Seção 4.4.2, formando assim, o *DataFrame*

inicial das empresas. Este procedimento foi realizado para todas as empresas que são analisadas neste trabalho.

Posteriormente, são analisadas as séries específicas de cada empresa, através de seus valores diários de abertura. O período coletado foi de 09/04/2001 a 31/08/2017.

Iniciando por ordem alfabética, as ações da Amazon contém os valores mais altos dentre as empresas estudadas. No período coletado, seu valor de abertura apresentou um mínimo de 5,91 e um máximo de 1069,55 dólares Americanos (USD), evidenciando seu crescimento e valorização. O Gráfico 2 representa o comportamento desta série.

GRÁFICO 2 – Valores de abertura das ações da Amazon
FONTE: Elaborado pelo autor

As ações da Apple contam com seus valores mais constantes, sem grande ascenção se comparada a Amazon. Seu valor de abertura apresentou um mínimo de 0,93 e um máximo de 163,80 USD. O Gráfico 3 representa o comportamento desta série.

As ações da Cisco, Intel e Microsoft possuem os valores mais equilibrados dentre as empresas selecionadas. O valor de abertura mínimo apresentado pela Cisco, no período coletado, foi de 8,45 enquanto seu valor máximo foi de 34,46 USD. Já a Intel, apresentou um valor um mínimo de 12,17 e um máximo de 38,25 USD. Por fim, a Microsoft também apresentou uma série bem equilibrada no período coletado, variando entre um valor mínimo de 15,20 e um máximo de 74,34 USD. Os Gráficos 4, 5 e 6 demonstram, de forma mais intuitiva, as ações da Cisco, Intel e Microsoft, respectivamente.

GRÁFICO 3 – Valores de abertura das ações da Apple
FONTE: Elaborado pelo autor

GRÁFICO 4 – Valores de abertura das ações da Cisco
FONTE: Elaborado pelo autor

GRÁFICO 5 – Valores de abertura das ações da Intel
FONTE: Elaborado pelo autor

GRÁFICO 6 – Valores de abertura das ações da Microsoft
FONTE: Elaborado pelo autor

5.2 EXPLORANDO O *DATAFRAME*

O refinamento do modelo de dados coletado é de suma importância para agregar valor às informações do *DataFrame* de cada empresa. Como especificado na Seção 4.3.1, é necessário calcular alguns indicadores técnicos, que são consideradas informações expressivas que influenciam diretamente nos valores das ações.

Para desenvolver o cálculo das médias móveis de 10 e 26 dias, foi formada uma equação levando em consideração o exposto na Seção 4.3.1, o que proporciona de uma forma mais clara como realizar a implementação do algoritmo. A equação (5.1) demonstra este procedimento:

$$M_t = \frac{Z_t + Z_{t-1} + \cdots + Z_{t-k+1}}{k}, \tag{5.1}$$

onde k é a quantidade de dias que se deseja calcular a média, t é o índice iterador da sequência que está sendo calculada e Z é o valor de fechamento da ação no momento t. A partir disto, foi desenvolvida uma função que realiza este cálculo. O Código 5 ilustra com detalhes esta implementação.

CÓDIGO 5 – Implementação da função que realiza o cálculo da média móvel

```
1 def executa_calculo_media_movel(self, indice_dataset, dia):
2     indice_dataset = indice_dataset + 1
3     indice_inicial = indice_dataset - (dia)
4     valor_media = 0
5     for i in range(indice_dataset):
6         if i >= indice_inicial:
7         valor_media = valor_media + self.dataset.loc[i].Close
8     self.dataset.set_value(indice_dataset - 1, 'movel_' + str(dia), ↩
    valor_media / dia, takeable=False)
9     self.dataset.to_csv('~/Documentos/TCC/dist-tcc/Implementacao/dados/'+↩
    self.nome_empresa+'_calculado.txt')
```

O Código 5 recebe como parâmetro o índice do *DataFrame* e a quantidade de dias que se deseja calcular a média móvel. O algoritmo encontra este índice e, a partir disso, faz um incremento com o valor de fechamento. Após atingir o limite de incrementos necessários, levando em consideração a quantidade de dias, realiza a divisão e salva o valor em uma nova coluna do *DataFrame*.

Já para o cálculo do indicador técnico MACD, o mesmo é realizado de forma mais simples com o auxílio da biblioteca pandas. O Código 6 cria uma nova coluna ao *DataFrame* e realiza o cálculo da diferença, utilizando a função "sub", entre as duas médias móveis.

CÓDIGO 6 – Implementação da função que realiza o cálculo do MACD

```
1 def calcula_macd(self, dataset):
2     dataset['MACD'] = dataset['movel_10'].sub(self.dataset['movel_26'])
3     return dataset
```

Finalizada a etapa de cálculo dos indicadores técnicos, todos os parâmetros necessários para iniciar a implementação da RNA, especificados no Capítulo 4, foram coletados e estruturados. A Figura 13 ilustra exatamente como ficou o *DataFrame* após a inserção das novas colunas.

```
id,Date,Open,High,Low,Close,Volume,movel_26,movel_10,MACD
0,2001-04-09,11.84,11.9,10.7,11.18,23281900,10.551923076900001,9.747,-0.8049230769000015
1,2001-04-10,11.24,13.5,11.2,12.01,18532400,10.5284615385,9.784,-0.7444615385000013
2,2001-04-11,13.1,13.75,12.18,13.32,13214100,10.5838461538,10.036,-0.5478461538000001
3,2001-04-12,13.05,15.01,13.0,14.67,11266000,10.676923076900001,10.503,-0.1739230769000013
4,2001-04-16,14.5,14.54,13.65,14.03,6126500,10.766923076900001,10.883,0.1160769230999986
5,2001-04-17,13.81,15.63,13.75,14.74,4840200,10.8626923077,11.447,0.5843076923000012
6,2001-04-18,15.66,18.16,15.55,16.54,19287400,11.0903846154,12.238,1.1476153846
7,2001-04-19,16.9,16.9,15.69,15.99,8337400,11.2823076923,12.997,1.7146923077
8,2001-04-20,15.62,16.2,14.97,15.78,8401000,11.4807692308,13.663,2.1822307692000003
9,2001-04-23,16.39,17.46,16.02,16.2,11567900,11.6853846154,14.446,2.7606153845999994
10,2001-04-24,16.44,17.62,15.48,15.68,6168400,11.8653846154,14.896,3.0306153845999986
11,2001-04-25,15.69,16.1,14.6,16.09,9234300,12.0803846154,15.304,3.2236153846000004
12,2001-04-26,16.3,16.5,15.0,15.43,5884000,12.2796153846,15.515,3.2353846154000014
13,2001-04-27,15.72,15.73,15.02,15.27,4497700,12.482307692300001,15.575,3.0926923076999984
14,2001-04-30,15.63,16.9,15.59,15.78,5479200,12.6973076923,15.75,3.0526923076999988
15,2001-05-01,15.9,17.05,15.8,16.89,4930300,12.955,15.965,3.01
16,2001-05-02,17.14,17.43,16.84,17.11,7437700,13.1923076923,16.022000000000002,2.8296923077000
17,2001-05-03,16.83,17.0,16.25,16.75,4962700,13.388846153800001,16.098,2.7091538461999978
18,2001-05-04,16.35,17.6,15.93,17.56,6122600,13.648846153800001,16.276,2.6271538461999993
19,2001-05-07,17.36,17.53,16.51,16.92,4983900,13.915,16.348,2.4330000000000003
20,2001-05-08,16.28,16.49,15.6,16.18,6290000,14.143846153800002,16.398,2.2541538461999977
21,2001-05-09,15.56,15.66,15.0,15.01,4604900,14.371153846199999,16.29,1.9188461538000008
22,2001-05-10,15.4,15.64,12.85,14.62,4832800,14.6015384615,16.209,1.607461538499999
23,2001-05-11,14.55,14.8,14.0,14.68,3341000,14.8430769231,16.15,1.3069230768999986
24,2001-05-14,14.53,14.53,13.15,13.33,6040300,15.005,15.905,0.8999999999999986
25,2001-05-15,13.35,14.3,13.14,13.54,6032800,15.2038461538,15.57,0.3661538461999996
26,2001-05-16,13.45,14.38,13.1,14.13,5678400,15.3173076923,15.272,-0.045307692299999765
27,2001-05-17,14.09,15.0,14.03,14.78,6220900,15.423846153800001,15.075,-0.3488461538000021
```

FIGURA 13 – Exemplo do *DataFrame* com os dados atualizados
FONTE: Elaborado pelo autor

5.3 NORMALIZANDO OS DADOS

Segundo Zárate, Helman e Gálvez (2003), o evento que antecede a etapa de treinamento de uma RNA é o processo de normalização dos dados de entrada e saída. Como o modelo proposto trabalha com a função de ativação sigmóide, detalhada na Seção 3.3.2, os dados devem ser normalizados entre um intervalo de [0,1]. A normalização consiste em adaptar uma base de dados com valores disintos, o que se aplica à realidade do presente trabalho, onde os valores das ações contam com intervalos de grande oscilação.

Considerando estas informações, duas equações foram especificadas para realizar a normalização e a desnormalização dos dados, com o objetivo de treinar a rede de forma mais eficaz, sendo elas:

$$L_n = (L_o - L_{min})/(L_{max} - L_{min}) \tag{5.2}$$

e

$$L_o = L_n * L_{max} + (1 - L_n) * L_{min}, \tag{5.3}$$

onde L_o é o valor à ser normalizado, L_n é o valor normalizado, L_{min} e L_{max} são os valores mínimos e máximos, respectivamente, dentre os valores da variável calculada.

A partir da equação (5.2), foi implementada uma função que normaliza uma determinada coluna de um *DataFrame*. O Código 7 detalha o desenvolvimento do método.

CÓDIGO 7 – Implementação da função de normalização

```
def normaliza_coluna(self, dataset, coluna):
    resultado = []
    for i in range(dataset.__len__()):
        resultado.append((dataset.iloc[i][coluna] - min(dataset[coluna]))↩
    / (max(dataset[coluna]) - min(dataset[coluna])))
    dataset[coluna + '-normalizado'] = resultado
    return dataset
```

Para efetuar a desnormalização de um determinado valor, levando em consideração a equação (5.3), foi implementada uma função que realiza este procedimento, detalhada no Código 8.

CÓDIGO 8 – Implementação da função de desnormalização de um valor

```
def desnormaliza_valor(self, dataset, coluna, valor):
    valor = valor * max(dataset[coluna]) + (1 - valor) * min(dataset[↩
    coluna])
    return valor
```

Tendo em vista o desenvolvimento dos *scripts* demonstrados, Código 7 e Código 8, foi utilizada a biblioteca matplotlib para gerar um gráfico e representar, com clareza, como ficaram os conjuntos de dados após o processo de normalização. O Código 9 detalha este procedimento.

CÓDIGO 9 – Classe que gera o gráfico dos dados normalizados

```
import pandas as pd
import matplotlib.pyplot as plt

class Normalizador:

    def __init__(self):
        try:
            self.dataset_cisco = pd.read_csv('~/Documentos/TCC/dist-tcc/↩
    Implementacao/dados_calculados/cisco_normalizado.txt')
            self.dataset_amazon = pd.read_csv('~/Documentos/TCC/dist-tcc/↩
    Implementacao/dados_calculados/amazon_normalizado.txt')
            self.dataset_microsoft = pd.read_csv('~/Documentos/TCC/dist-↩
    tcc/Implementacao/dados_calculados/microsoft_normalizado.txt')
            self.dataset_intel = pd.read_csv('~/Documentos/TCC/dist-tcc/↩
    Implementacao/dados_calculados/intel_normalizado.txt')
            self.dataset_apple = pd.read_csv('~/Documentos/TCC/dist-tcc/↩
    Implementacao/dados_calculados/apple_normalizado.txt')
        except IOError:
            print("Erro ao abrir os dados das empresas")
            return 0

    def define_linha_coluna(self):
```

```
18          self.linha = self.dataset_cisco['Date'] = pd.to_datetime(self.↩
    dataset_cisco['Date'])
19          self.coluna = self.dataset_cisco['Open-normalizado']
20          self.coluna1 = self.dataset_amazon['Open-normalizado']
21          self.coluna2 = self.dataset_microsoft['Open-normalizado']
22          self.coluna3 = self.dataset_intel['Open-normalizado']
23          self.coluna4 = self.dataset_apple['Open-normalizado']
24
25      def cria_grafico(self):
26          fig, ax = plt.subplots()
27          ax.plot(self.linha, self.coluna)
28          ax.plot(self.linha, self.coluna1)
29          ax.plot(self.linha, self.coluna2)
30          ax.plot(self.linha, self.coluna3)
31          ax.plot(self.linha, self.coluna4)
32          fig.set_size_inches(12, 8, forward=True)
33          plt.legend(['Cisco', 'Amazon', 'Microsoft', 'Intel', 'Apple'])
34          plt.grid(True)
35          plt.show()
36          plt.close()
```

Após a execução do Código 9, foi renderizado o Gráfico 7 que demonstra os valores de cada empresa distribuidos entre o intervalo [0,1].

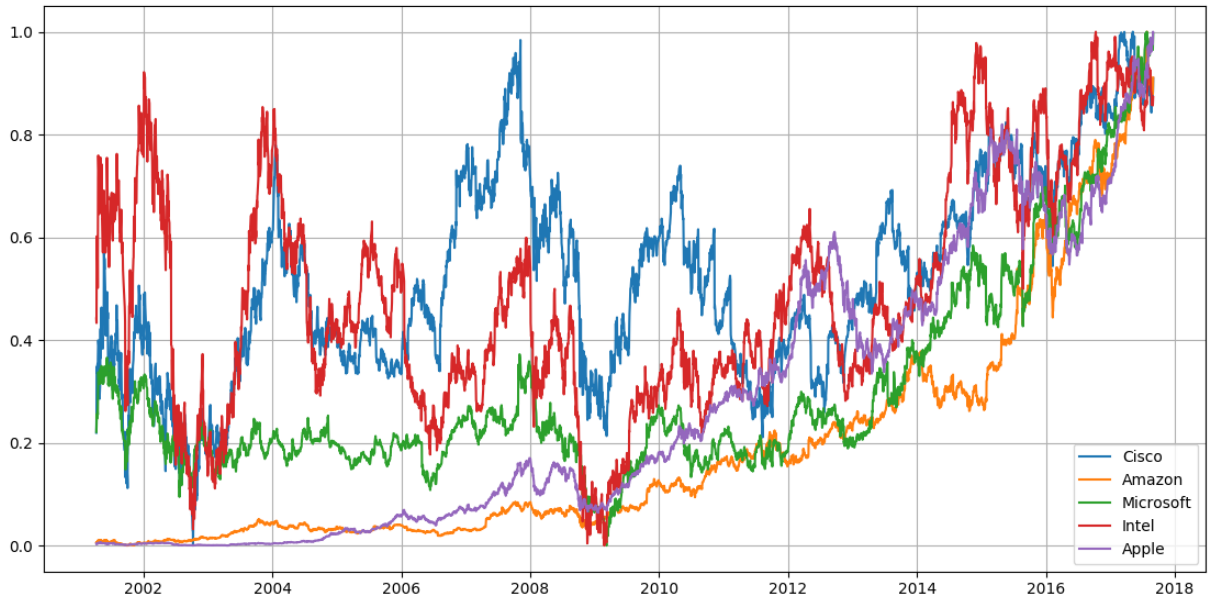


GRÁFICO 7 – Dados normalizados para treinamento
FONTE: Elaborado pelo autor

5.4 DESENVOLVIMENTO DA REDE NEURAL ARTIFICIAL

O ambiente de desenvolvimento do modelo de RNA proposto baseou-se nas ferramentas definidas no capítulo anterior, sendo elas a linguagem de programação Python, em sua versão 3.6.0, e a biblioteca PyBrain. Esta Seção detalha todos os procedimentos realizados para a implementação do modelo desenvolvido, evidenciando aspectos gerais de como trabalhar com Python e PyBrain.

5.4.1 *Script* para criação de um modelo *FeedForward*

A implementação de uma RNA com a topologia *FeedForward* foi construída através do módulo pybrain.structure.networks. O mesmo tem como objetivo, especificar como são tratados o fluxo dos dados de entrada da rede. O Código 10 demonstra a importação do módulo e a criação da topologia determinada.

CÓDIGO 10 – Construção de uma RNA com a estrutura *FeedForward*

```
1 from pybrain.structure import FeedForwardNetwork
2
3 class FeedForwardNetworkPyBrain:
4
5     def __init__(self):
6         self.rede = FeedForwardNetwork()
```

5.4.2 Organização das camadas do modelo

Após a implementação da topologia, faz-se necessário organizar suas camadas, assim como são tratados os respectivos neurônios das mesmas. Para a construção desta etapa, a biblioteca PyBrain oferece, através do módulo pybrain.structure, suporte à inserção via parâmetros, especificando a quantidade de variáveis para cada camada e, também, suas características. No Código 11 é possível observar a classe que implementa o método desenvolvido para a construção das camadas.

CÓDIGO 11 – Implementação da classe de criação de camadas

```
1 from pybrain.structure import LinearLayer, SigmoidLayer
2
3 class FeedForwardNetworkPyBrainLayers:
4
5     def __init__(self, tamanho_camada_entrada, tamanho_camada_oculta, ↩
    tamanho_camada_saida):
6         self.camada_entrada = LinearLayer(self.camada_entrada, name="↩
    entrada")
7         self.camada_oculta = SigmoidLayer(self.camada_oculta, name="↩
    oculta")
8         self.camada_saida = LinearLayer(self.camada_saida, name="saida")
```

Neste código são importados dois objetos do tipo *Layer*, denominados *LinearLayer* e *SigmoidLayer*. A camada de entrada é composta pelo objeto *LinearLayer*, onde os dados não sofrem nenhuma transformação, apenas transferindo-os para a camada posterior. A camada oculta é composta pelo objeto *SigmoidLayer*, que implementa o cálculo da Equação (3.4) para os dados recebidos. Por fim, a camada oculta também é composta pelo objeto *LinearLayer*, resultando na resposta da rede.

Realizada a etapa de especificação das camadas e suas respectivas funcionalidades, é necessário adicioná-las, efetivamente, ao modelo de rede. Este procedimento deve ser feito por meio dos métodos *addInputModule*, *addModule* e *addOutputModule*. O Código 12 demonstra a função desenvolvida para a inserção das camadas.

CÓDIGO 12 – Inserção de camadas em um modelo de RNA

```
1 def adicionaEstrutura(self, rede):
2     rede.addInputModule(self.camada_entrada)
3     rede.addModule(self.camada_oculta)
4     rede.addOutputModule(self.camada_saida)
5     return rede
```

Observando este código, pode-se notar que o mesmo recebe um modelo de rede como parâmetro, faz a inserção através dos respectivos métodos citados anteriormente e retorna a rede com a nova configuração.

A partir disto, a rede precisa identificar quais são as distribuições de conexões referente às camadas inseridas, para determinar o fluxo de trabalho dos dados. Este procedimento é realizado através do objeto *FullConnection*, que determina a ligação entre as camadas. Como o modelo será composto por 3 camadas, faz-se necessário realizar duas conexões, uma entre a camada de entrada e a camada oculta e outra entre a camada oculta e a camada de saída. No Código 13 é demonstrado o método desenvolvido para criar esta funcionalidade.

CÓDIGO 13 – Conectando as camadas criadas na RNA

```
1 def adicionaConexoes(self):
2     from pybrain.structure import FullConnection
3     '''Importação do objeto FullConnection através do módulo pybrain.↩
    structure '''
4     self.ligacao_entrada_oculta = FullConnection(self.camada_entrada, ↩
    self.camada_oculta)
5     self.ligacao_oculta_saida = FullConnection(self.camada_oculta, self.↩
    camada_saida)
6     self.network.addConnection(self.ligacao_oculta_saida)
7     self.network.addConnection(self.ligacao_entrada_oculta)
```

Concluída a fase de inserção das conexões, apresentada no Código 13, faz-se necessário inserir os pesos iniciais da rede. Este procedimento é executado através do método

sortModules, disponbilizado pela biblioteca PyBrain, que adiciona, de forma aleatória, os pesos em um intervalo [-2,2].

5.4.3 Treinamento da rede

Para o desenvolvimento do treinamento a biblioteca PyBrain utiliza dois módulos principais, sendo eles o pybrain.datasets, que é responsável por especificar qual o modelo de aprendizado que a rede trabalha, e, para o caso do aprendizado supervisionado, o módulo pybrain.supervised, que contém os respectivos algoritmos para a presente técnica.

Para mapear uma base de dados de aprendizagem supervisionada com as características desejadas, é necessário utilizar o método *SupervisedDataSet*, onde o mesmo recebe como parâmetro a quantidade de entradas referente à base de dados utilizada e a quantidade de saídas desejadas. Para o presente trabalho, foi instanciada uma base de dados de treino para 8 entradas e 1 saída, seguindo as especificações da Seção 4.5.1.

O Código 14 expõe o desenvolvimento do método que realiza a criação da base de treino.

CÓDIGO 14 – Construção de uma base de dados para treinamento

```
def adicionaDadosTreinamento(self, nome_empresa):
    import pandas as pd
    from pybrain.datasets import SupervisedDataSet

    try:
        self.dataset = pd.read_csv('~/Documentos/TCC/dist-tcc/Implementacao/dados_calculados/'+ nome_empresa + '_normalizado.txt',header=0)
    except IOError:
        print ("Erro ao abrir os dados da empresa "+nome_empresa+"")
        return 0

    self.dataset_treino = SupervisedDataSet(8, 1)

    for i in range(self.dataset.__len__() - 8):
        self.dataset_treino.addSample([
        self.dataset.iloc[i]['Open-normalizado'],
        self.dataset.iloc[i]['High-normalizado'],
        self.dataset.iloc[i]['Low-normalizado'],
        self.dataset.iloc[i]['Close-normalizado'],
        self.dataset.iloc[i]['Volume-normalizado'],
        self.dataset.iloc[i]['movel_26-normalizado'],
        self.dataset.iloc[i]['movel_10-normalizado'],
        self.dataset.iloc[i]['MACD-normalizado']],
        self.dataset.iloc[i + 1]['Open-normalizado'])

    return self.dataset_treino
```

Observando este código, pode-se notar que as linhas 2 e 3 são responsáveis por importar as dependências das bibliotecas utilizadas. Entre as linhas 5 e 9, é realizada a leitura do arquivo da empresa que foi transferida para a função via parâmetro (nome_-empresa). A linha 11 cria o objeto *SupervisedDataSet* com suas devidas configurações, enquanto a linha 13 consome a base de treino a partir do *DataFrame* sem seus oito últimos dias, que são utilizados para testes. Já a linha 14 inicia o processo de inserção das variáveis do *DataFrame* para a nova base, através do método *addSample*.

Também é possível notar, em relação ao *DataFrame*, que são utilizados os dados normalizados para consumir a base de treinamento. Por fim, a distribuição da camada de saída, encontrada na linha 23, está sendo carregada pelo valor de abertura no dia posterior ao estudado, pelo índice [i + 1], caracterizando, de forma correta, a base de treino para o problema proposto.

Após concluído os procedimentos anteriores, pode-se iniciar, efetivamente, o treinamento da RNA. É importante ressaltar que, neste trabalho, o treino foi executado através do algoritmo *Backpropagation*. O Código 15 demostra, de forma simples, como trabalhar com o algoritmo através da biblioteca PyBrain.

CÓDIGO 15 – Método de treinamento da RNA através do *Backpropagation*

```
1 def realizaTreinamento(self, rede, dataset_treino, epocas, nome_empresa):
2     from pybrain.supervised import BackpropTrainer
3     from pybrain.tools.customxml import NetworkWriter
4
5     treinamento = BackpropTrainer(rede, dataset_treino, learningrate=0.4,↩
      verbose=True)
6     treinamento.trainEpochs(epochs = epocas)
7     NetworkWriter.writeToFile(rede, 'snapshot_redes/rede-feedforward-' + ↩
     nome_empresa + '.xml')
```

A função implementada no Código 15, inicia o processo de treino a partir da chamada ao objeto *BackPropTrainer*. Este, por sua vez, recebe alguns parâmetros no momento de sua criação, sendo eles: a rede neural em sí, a base de treinamento e a taxa de aprendizagem (*learning rate*), detalhada na Seção 3.5.1. Segundo Haykin (2000), um valor recomendado para o parâmetro *learning rate* é de 0.4 (40%). No presente trabalho, foi utilizada esta taxa como referência para o treinamento, evidenciada na linha 5 do Código 15. Já o parâmetro *verbose*, auxilia o acompanhamento do erro quadrático médio durante o processo de treinamento, guardando as respectivas variações em um vetor unidimensional.

A partir disto, faz-se necessário realizar a chamada ao método *train*. O mesmo recebe um valor responsável por limitar a quantidade de ciclos (iterações) na qual o modelo é treinado, através da variável *epochs*.

O objeto *NetworkWriter*, encontrado na linha 7, pertence ao pacote tools.customxml, implementando pela biblioteca, que utiliza uma classe que pode receber o estado atual

de uma rede treinada e gravá-la em um arquivo *eXtensible Markup Language* (XML). Segundo Paoli et al. (2004), o XML foi criado para gerar linguagens de notação para necessidades especiais, de fácil manuseio e portabilidade de sua estrutura. Portanto, tendo em vista a capacidade de exportação da RNA para um arquivo XML, algumas possibilidades são oferecidas, tais como: retreinar a mesma, de forma simplificada, e testá-la quando desejado, sem a necessidade de realizar o processo de treinamento novamente.

A estrutura do arquivo gerado, levando em consideração a exportação realizada pelo Código 15, é demonstrada detalhadamente no Código 16.

CÓDIGO 16 – Estrutura do arquivo XML de uma RNA

```
<?xml version="1.0" ?>
<PyBrain>
  <Network class="pybrain.structure.networks.feedforward.FeedForwardNetwork" name="FeedForwardNetwork-5">
    <name val="'FeedForwardNetwork-5'"/>
    <Modules>
      <LinearLayer class="pybrain.structure.modules.linearlayer.LinearLayer" inmodule="True" name="entrada">
        <name val="'entrada'"/>
        <dim val="8"/>
      </LinearLayer>
      <LinearLayer class="pybrain.structure.modules.linearlayer.LinearLayer" name="saida" outmodule="True">
        <name val="'saida'"/>
        <dim val="1"/>
      </LinearLayer>
      <SigmoidLayer class="pybrain.structure.modules.sigmoidlayer.SigmoidLayer" name="oculta">
        <name val="'oculta'"/>
        <dim val="13"/>
      </SigmoidLayer>
    </Modules>
    <Connections>
      <FullConnection class="pybrain.structure.connections.full.FullConnection" name="FullConnection-4">
        <inmod val="entrada"/>
        <outmod val="oculta"/>
        <Parameters>[-1.5391790734038937, -0.49055463785006825, 1.0086336433363452, -2.2174173142083884, 0.34022915664783759, 0.81368148646844307, -0.79312561710231066, -0.2064521255668551, 0.6143776143608024, 0.14342628566708296, -0.83340451988100794, -0.36956268604968023, -0.54586351715424564, 0.049980197948936246, -0.83076432851705551, -0.55618633885483804, -0.18832466217890553, 0.78933557683280886, 1.316156433586583, 0.65498058923629887, -0.095088165246523901, -0.072001187710360778, -1.0633653587732319, -0.82076982729096515, 1.3591603265398959, 1.3133540634261023,
```

```
    −1.8140666281670779, −0.6923749694990321, 0.64241794608246172, ↩
    0.085592998908258361, 0.68871476531735654, −1.2620647896756734, ↩
    −2.3737144705981641, −1.24468183407378, −0.45967805715659982, ↩
    1.3668566718717532, −1.371367463098293, −0.43622700918061857, ↩
    −1.077322847095302, 0.019845048918964058, −0.23688807362421782, ↩
    −0.98422983819481236, 0.17676833784966017, 0.54709854709655015, ↩
    −0.17491065539440157, −0.89157076543318903, 0.22294392160022539, ↩
    −1.1272830485683025, −2.2946043337978548, 0.70302705521542697, ↩
    −1.8273112913922722, 0.72521317536403651, −1.281056549296673, ↩
    −0.91857441417523766, 1.0467188711344912, −1.0327391547503506, ↩
    1.2253972832244919, 0.45778530014513541, 1.5413191896375917, ↩
    −3.8834704934401358, 0.57737056085416039, 0.31567210721378625, ↩
    −0.85529464757490603, −0.70133113550434778, −0.017148679131095825, ↩
    0.2876441369265601, 0.73812213113724789, −0.4049583153198219, ↩
    0.56336248222969532, −1.0205618509408894, −0.92979753952865341, ↩
    −0.81240872214453841, 1.252634269719118, −0.56776850292197956, ↩
    0.21143942856188347, 1.7466136963396783, −0.40437677920167114, ↩
    −0.61375509642007031, −0.98285719161773677, −0.13405686951183735, ↩
    −1.295175666199176, −1.0629196040206275, 2.3552510307718246, ↩
    0.27057980229962858, 1.6019067516339633, −1.7377959799800631, ↩
    −1.0640131144569442, −1.5711735485701748, −0.29631524832813327, ↩
    −0.48477188618894684, −1.5721560034296649, −1.8701154538117915, ↩
    −0.11021543882141212, 0.59393123606042808, 0.12058383654737018, ↩
    0.20141130497360898, 0.84337932265436288, 0.59812610513048903, ↩
    −0.81727861722487327, −0.82410244903076124, −0.28995338622948574, ↩
    0.81216674317365312, −1.7809071435855139, −2.1112897151201304]</↩
    Parameters>
24      </FullConnection>
25      <FullConnection class="pybrain.structure.connections.full.↩
    FullConnection" name="FullConnection-3">
26          <inmod val="oculta"/>
27          <outmod val="saida"/>
28          <Parameters>[0.82703637888584824, −1.0993317566130805, ↩
    0.85155184453785016, 0.39205349930277422, −0.19792323388926678, ↩
    −0.59635484999519217, 0.29807705379491239, −0.70833834579653443, ↩
    −0.64179050961098394, 0.84805052376391954, 0.12824478317036711, ↩
    −0.2092038898936153, 0.046612816278560475]</Parameters>
29      </FullConnection>
30    </Connections>
31  </Network>
32 </PyBrain>
```

Observando o Código 16, pode-se notar que todas as configurações executadas para construir a rede são armazenadas dentro de suas respectivas características, denominadas *tags*. As *tags* são padronizadas para que, no momento de leitura da RNA, através do objeto *networkReader*, todo o ambiente seja importado de forma correta. Analisando as linhas 3, 6 e 14, por exemplo, é possível notar alguns dos parâmetros inseridos anteriormente na

RNA. Além disso, também fica evidente a presença dos pesos sinápticos, entre a camada de entrada e a camada oculta, e, entre a camada oculta e a camada de saída, que são fundamentais para salvar um determinado estado da rede.

Por fim, tendo em vista o refinamento de todas as bases de dados coletadas e da implementação do modelo da RNA, todos os procedimentos necessários para realizar os testes foram desenvolvidos, com base na teoria e especificações dos capítulos anteriores.

6 ANÁLISE DOS RESULTADOS

Este capítulo tem como finalidade apresentar os resultados obtidos através das implementações demonstradas no Capítulo 5.

6.1 ANÁLISE DAS SÉRIES UTILIZADAS

Após a implementação do modelo da RNA, através dos conjuntos de dados coletados, faz-se necessário avaliar como a respectiva técnica se comportou. Tendo isso em vista, a mesma foi aplicada em função do objetivo principal do trabalho, que é medir a capacidade de precisão de acerto no valor de abertura das ações. Portanto, foi elaborada, de forma independente, uma análise de eficácia do modelo para o cenário de cada empresa utilizada no presente trabalho. É importante ressaltar que os modelos foram implementados levando em consideração todos os *scripts*, métodos e ferramentas utilizadas no capítulo anterior.

6.1.1 Aplicação da rede na Intel Corporation

A rede da Intel Corporation foi treinada com o objetivo de capturar o maior nível de variação possível dos dados, visando mapear um maior conjunto de padrões e, assim, responder de forma eficiente à dados dispersos através de uma boa capacidade de generalização. Tendo isso em vista, o período de treinamento preparado foi de 09/04/2001 até 21/08/2017, totalizando 4117 registros.

Para o presente caso foram construídos dois cenários de simulação, buscando obter um parâmetro de ciclos de treinamento adequado ao modelo de dados utilizado. As métricas definidas para análise, a partir destes ciclos de treinamento, são: o comportamento da função de custo que compõem o modelo (erro quadrático médio, EQM) e a margem de erro dos valores resultantes da rede, no período de 23/08/2017 a 31/08/2017, em relação aos valores reais. Os parâmetros utilizados foram: 200 e 1000 iterações.

6.1.1.1 Treinamento com 200 Iterações

A ideia de executar o treinamento com uma quantidade baixa de iterações, é feita com o intuito de proporcionar um treinamento mais rápido e com resultados significativos, levando em consideração as características especificas do modelo de dados. Portanto, para quantificar como ocorreu o processo de treinamento, basta multiplicar o número de iterações (200) com o número de registros de testes (4117), resultando em um total de 823.400 exemplos calculados pela rede. O Gráfico 8 demonstra, de acordo com a quantidade de ciclos, a variação do EQM no primeiro cenário de teste com 200 iterações.

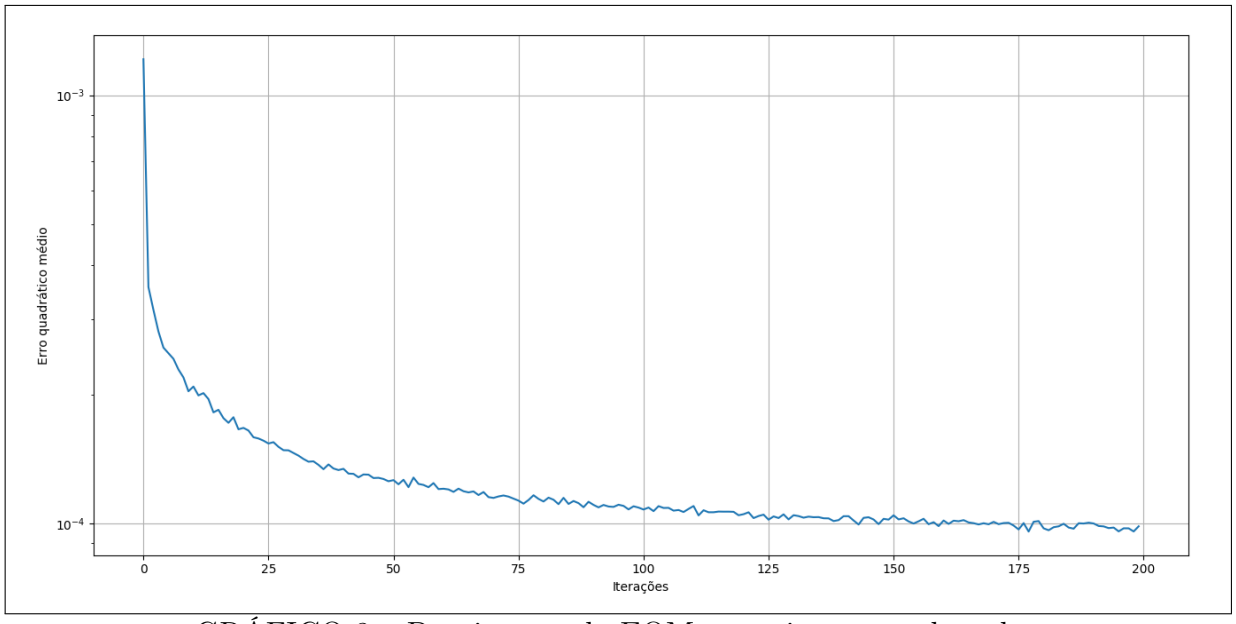


GRÁFICO 8 – Decaimento do EQM no treinamento da rede
FONTE: Elaborado pelo autor

O EQM de treinamento iniciou-se, na primeira iteração, com um erro percentual de 0.00248996652176 e, após o término das 200 épocas de treinamento, conclui-se com um erro percentual de $9.44359025338.10^{-05}$. Analisando o gráfico, pode-se observar que os valores tiveram um decaimento constante até a iteração de número 125, enquanto que, nas 75 iterações posteriores, manteve os valores dos erros aproximados. A queda constante nas iterações iniciais está diretamente ligada a capacidade de aprendizado e adaptação ao modelo de dados.

Tendo em vista a variação dos erros obtidos após a primeira inicialização da rede, foi julgado necessário repetir o mesmo procedimento, visando observar o funcionamento do algoritmo de inicialização dos pesos e validar sua variação aplicada ao mesmo modelo de dados. Portanto, mais dois cenários foram executados com este objetivo. No segundo cenário, o erro foi iniciado com um percentual de 0.00102981131422 e, após o término das 200 épocas de treinamento, concluí-se com um percentual de $9.1790170623.10^{-05}$. No terceiro cenário, o erro foi iniciado com um percentual de 0.00026991101482 e, após o término das 200 épocas de treinamento, concluí-se com um percentual de $9.49959768553.10^{-05}$.

Analisando os três cenários, pode-se observar que apesar da rede ser iniciada com os pesos aleatoriamente, o EQM seguiu a mesma tendência para todos os casos, isso implica em uma confiabilidade maior por parte do algoritmo, pois o mesmo garante que novas inicializações, aplicadas ao mesmo modelo de dados, não resultam em valores muito distintos. Esta validação do algoritmo tem grande importância, pois com ela chega-se a definição de que os pesos iniciais da rede não precisarão, necessariamente, serem ajustados manualmente para encontrar um melhor resultado, pois aplicado ao mesmo modelo de dados, os erros seguem o mesmo padrão.

Após a análise do comportamento do EQM e a validação do algortimo de inicialização, os testes foram iniciados com o objetivo de verificar a precisão do modelo. Na Tabela 1 são detalhados os valores do período predito.

TABELA 1 – Período dos dados utilizados para testes: Intel Corporation

Data	Abertura	Alta	Baixa	Fechamento	Volume
23/08/2017	34.54	34.81	34.38	34.66	196.481,34
24/08/2017	34.70	34.89	34.55	34.71	143.018,92
25/08/2017	34.82	34.93	34.58	34.67	147.268,29
28/08/2017	34.78	34.80	34.59	34.65	207.128,76
29/08/2017	34.51	34.75	34.46	34.73	158.436,68
30/08/2017	34.75	34.96	34.63	34.89	185.650,07
31/08/2017	34.94	35.18	34.87	35.07	163.667,72

Os dados demonstrados na Tabela 1 foram refinados e normalizados de acordo com os métodos implementados no Capítulo 5. Após a execução deste processo, os mesmos foram inseridos na RNA para ativação. Posteriormente, foi realizada a desnormalização e os resultados apresentados. A Tabela 2 ilustra os resultados obtidos.

TABELA 2 – Resultados da predição realizada nos dados utilizados pela rede

Data	Valor real	Resultado	Erro (%)	Variação
23/08/2017	34.54	34.73	0.550	-0.19
24/08/2017	34.70	34.71	0.028	-0.01
25/08/2017	34.82	34.79	0.086	0.03
28/08/2017	34.78	34.77	0.028	0.01
29/08/2017	34.51	34.75	0.695	-0.24
30/08/2017	34.75	34.82	0.201	-0.07
31/08/2017	34.94	34.99	0.143	0.05

Analisando a Tabela 2, pode-se observar que os resultados obtidos foram significativos, onde o percentual de erro calculado, através do erro relativo percentual, não ultrapassou a margem 0.70%, se aproximando, consideravelmente, dos valores reais. O erro médio, analisado para todo o período de predição, ficou em torno de 0.24%, o que pode ser considerado baixo levando-se em conta o número de iterações utilizadas para treinar a RNA. Já a medida de dispersão dos erros em torno da média obtida (desvio padrão) foi de aproximadamente 0.26%. Entretanto, analisando de forma mais detalhada cada valor resultante, pode-se evidenciar dois erros mais altos nos dias 23/08/2017 e 29/08/2017, com 0.550% e 0.695%, respectivamente. Tendo em vista que os valores mais próximos aos verdadeiros oscilaram no intervalo entre [34.70, 34.82], fica evidente que o modelo obteve erros mais altos em momentos de queda no valor desta série. A ocorrência destes erros

acontece através da tendência de padrões em que a rede foi treinada, de forma crescente, não acompanhando assim as quedas bruscas no período. Isto fica mais claro observando o valor predito do dia 31/08/2017, onde o valor real obteve uma alta considerável, se comparada aos valores anteriores, porém não afetou a capacidade de precisão da RNA, mantendo a margem de erro pequena. O Gráfico 9 representa, de maneira ilustrativa, os resultados da série.

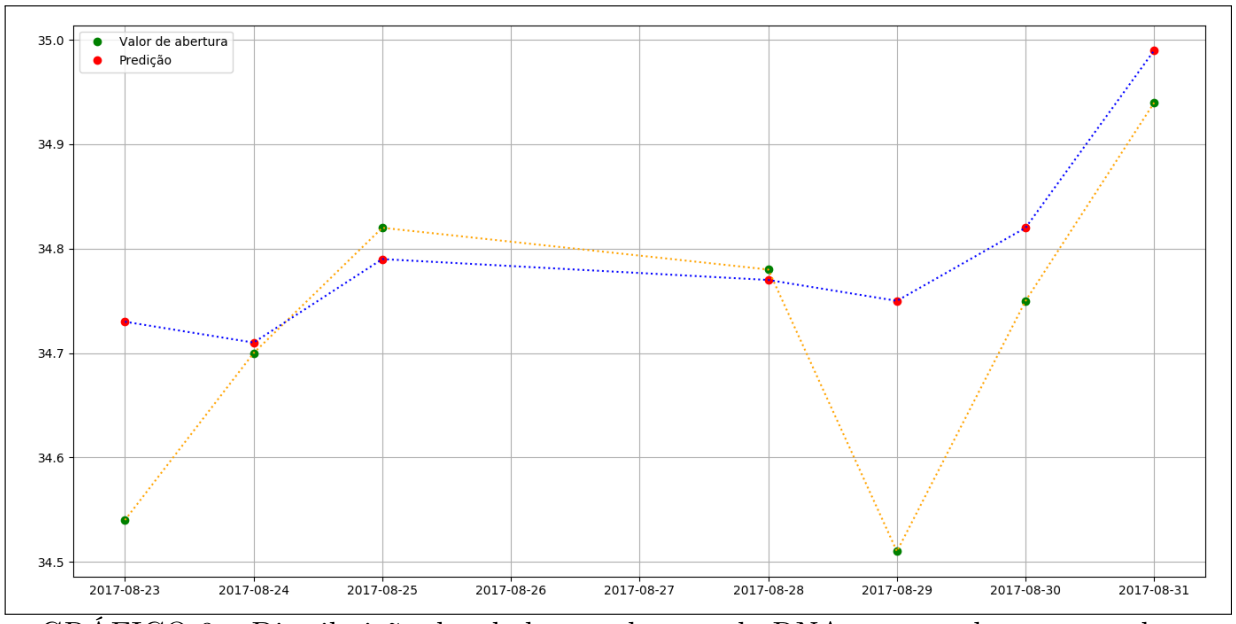


GRÁFICO 9 – Distribuição dos dados resultantes da RNA e seus valores esperados
FONTE: Elaborado pelo autor

Analisando o gráfico é possível observar como os resultados são próximos aos esperados. Os valores de abertura, nos respectivos dias testados, são representados por um ponto verde. Já os resultados obtidos pela rede são caracterizados pelo ponto vermelho. No Gráfico também fica evidente, de maneira ilustrativa, como a série não acompanhou as quedas do período e seguiu uma tendência crescente.

6.1.1.2 Treinamento com 1000 Iterações

Com a finalidade de comparar com a simulação anterior, foi realizado um teste com 1000 épocas de treinamento visando alcançar um resultado mais preciso e, também, para validar se o aumento na quantidade de iterações, no processo de treinamento, melhoram os resultados retornados pela rede.

Para quantificar como ocorreu o processo de treinamento, basta multiplicar o número de iterações (1000) com o número de registros de testes (4117), resultando em um total de 4.117.000 exemplos calculados pela rede.

Para o primeiro cenário a rede iniciou-se, na primeira iteração, com um erro percentual de 0.00664064049629 e, após o término das 1000 épocas de treinamento, conclui-se com um erro percentual de $8.7943563121853531.10^{-05}$. Mais dois treinos distintos foram

realizados visando encontrar alguma dispersão no erro obtido pela rede, como feito na análise anterior. Portanto, o segundo treino obteve um erro inicial de 0.00260377914122 e, após o término das 1000 épocas de treinamento, conclui-se com um percentual de $8.91402403614.10^{-05}$. O terceiro treino obteve um erro inicial de 0.00496238079711 e, após o término das 1000 épocas de treinamento, o EQM do treinamento conclui-se com um percentual de $8.90814371631.10^{-05}$.

Após a execução desses cenários de treinamento, pode-se concluir que a rede se comportou de forma estável em relação a quantidade de iterações, não distinguindo, de forma relevante, os valores dos pesos em diversas inicializações aleatórias. O Gráfico 10 representa, de maneira ilustrativa, o comportamento do EQM da rede treinada pelo primeiro cenário.

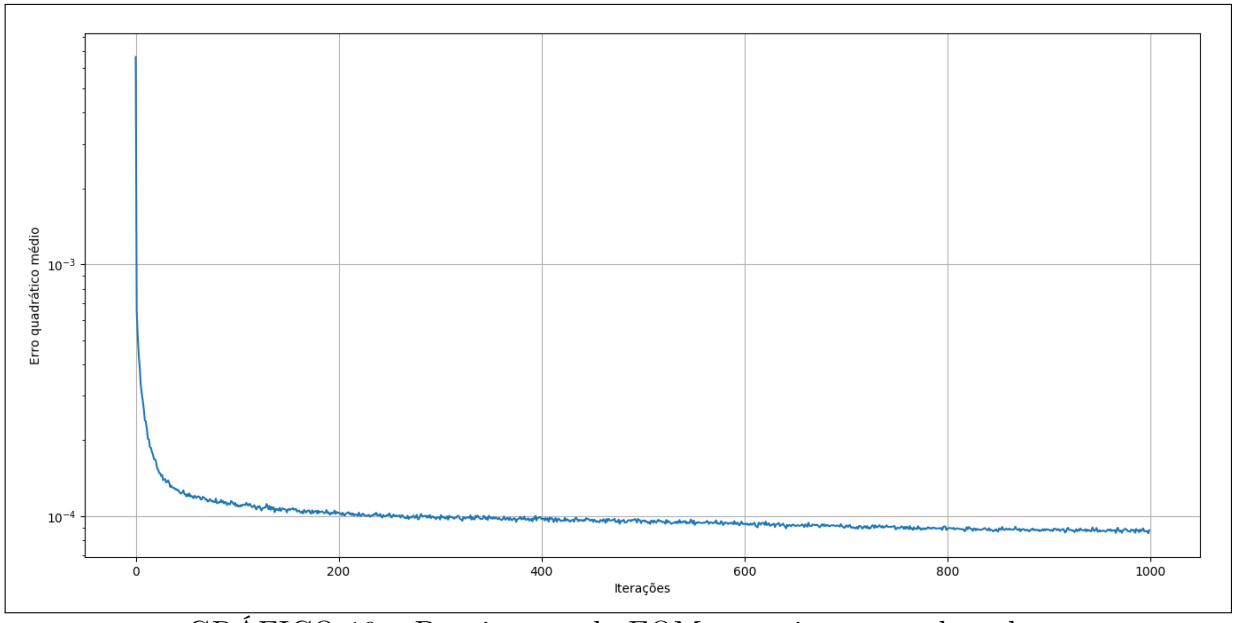


GRÁFICO 10 – Decaimento do EQM no treinamento da rede
FONTE: Elaborado pelo autor

Analisando o Gráfico 10, pode-se observar que o erro obteve uma queda constante até 200 iterações, enquanto que, entre as iterações de número 200 até a de número 400, manteve o padrão com valores aproximados. Os últimos 600 ciclos mantiveram um decaimento mínimo do EQM.

Tendo em vista a simulação anterior, com 200 iterações, fica claro que a queda brusca e rápida aconteceria nas primeiras iterações, pois a base de dados utilizada é a mesma. Também é possível observar, através de um número maior de iterações, que o EQM obteve resultados mais baixos em comparação ao treino anterior.

O período aplicado para a realização dos testes foi o mesmo utilizado da Tabela 1. Os dados foram devidamente refinados, normalizados e, a partir disto, foram ativados na rede. Os resultados preditos, a partir da execução dos testes, são demonstrados na Tabela 3.

TABELA 3 – Resultados da predição realizada nos dados utilizados pela rede

Data	Valor real	Resultado	Erro (%)	Variação
23/08/2017	34.54	34.67	0.376	-0.13
24/08/2017	34.70	34.64	0.172	0.06
25/08/2017	34.82	34.71	0.315	0.11
28/08/2017	34.78	34.68	0.287	0.10
29/08/2017	34.51	34.65	0.405	-0.14
30/08/2017	34.75	34.70	0.143	0.05
31/08/2017	34.94	34.84	0.286	0.10

Analisando a Tabela 3, pode-se observar que os resultados obtidos foram significativos, onde o percentual de erro calculado, através do erro relativo percentual, não ultrapassou a margem 0.50%, se aproximando consideravelmente dos valores reais. Porém, a média de todo o período analisado obteve um erro de 0.28%, superior ao treinamento anterior. Já a medida de dispersão dos erros em torno da média obtida (desvio padrão) foi de aproximadamente 0.09%. Sendo assim, pode-se concluir que o desvio padrão foi menor que o do primeiro cenário, devido ao fato dos erros mais altos (23/08/2017 e 29/08/2017) serem mais suaves. O Gráfico 11 representa, de maneira ilustrativa, os resultados da rede.

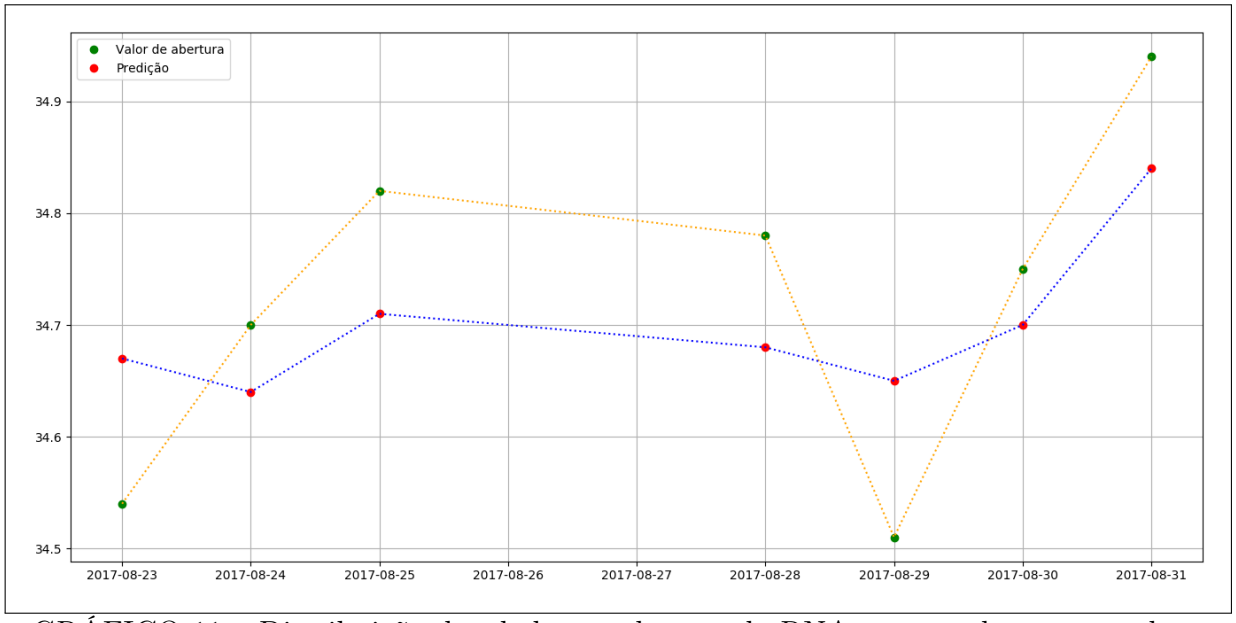


GRÁFICO 11 – Distribuição dos dados resultantes da RNA e seus valores esperados
FONTE: Elaborado pelo autor

Apesar da diferença do erro percentual relativo entre os dois cenários ser de apenas 4%, foi encontrada uma melhor parâmetrização para o presente modelo de dados utilizando um treinamento com 200 iterações, pois a exatidão dos valores preditos foram mais eficazes neste método de treinamento. Sendo assim, chega-se a definição que nem sempre um número maior de ciclos de treinamento aumentam a precisão da RNA. Neste caso, a rede sofreu uma convergência do EQM em torno de 200 iteraçoes, este detalhe pode

ser observado visualizando o comportamento do Gráfico 10. Dessa forma, as iterações posteriores propagaram o erro de forma desnecessária, alterando assim os pesos sinápticos dos neurônios e, automaticamente, causando uma maior dispersão nos resultados. Este processo ocorrido na segunda simulação, na literatura, é denominado *Overtraining*, detalhado no Capítulo 3.

6.1.2 Aplicação da rede na Microsoft Corporation

A rede da Microsoft Corporation foi treinada utilizando as mesmas técnicas da Intel Corporation, ou seja, com o intuito de capturar o maior nível de variação possível dos dados, visando mapear um maior conjunto de padrões e, assim, obter uma boa capacidade de generalização. Tendo isso em vista, o período de treinamento preparado foi de 09/04/2001 até 21/08/2017, totalizando 4117 registros.

Para o presente caso foram construídos dois cenários de simulação, buscando obter um parâmetro de ciclos de treinamento adequado ao modelo de dados utilizado. As métricas definidas para análise, a partir destes ciclos de treinamento, são: o comportamento da função de custo que compõem o modelo (erro quadrático médio, EQM) e a margem de erro dos valores resultantes da rede, no período de 23/08/2017 a 31/08/2017, em relação aos valores reais. Os parâmetros de ciclos de treinamento utilizados foram: 200 e 1000 iterações.

6.1.2.1 Treinamento com 200 Iterações

O treinamento com 200 iterações foi executado com o objetivo de verificar a capacidade de precisão da rede com um número menor de iterações. Para quantificar como ocorreu o processo de treinamento, basta multiplicar o número de iterações (200) com o número de registros de testes (4117), resultando em um total de 823.400 exemplos calculados pela rede.

Para este caso, a RNA foi inicializada duas vezes com o objetivo de observar o comportamento e a tendência do EQM em relação ao algoritmo de inicialização. No primeiro cenário, a rede iniciou-se com o EQM em 0.0018579672451418453 e, após as 200 iterações, conclui-se com um erro de $3.6623220378015219.10^{-05}$. No segundo cenário, o erro foi inicializado com 0.0047598765727053707 e, após as 200 iterações, conclui-se com um erro de $3.8483854181673219.10^{-05}$.

É possível verificar, através dos dois cenários executados, que o erro seguiu a mesma tendência para os dois casos, com um erro final próximo a 3.10^{-05}. Tendo isso em vista, possíveis inicializações distintas aplicada ao modelo de dados da Microsoft Corporation não irão resultar em valores muito distintos, pois a diferença do erro é mínima.

Para demonstrar, de maneira mais clara, o comportamento do erro obtido pela rede em todo processo de treinamento, o Gráfico 12 demonstra, de acordo com a quantidade

ciclos, a variação do EQM no primeiro cenário de teste.

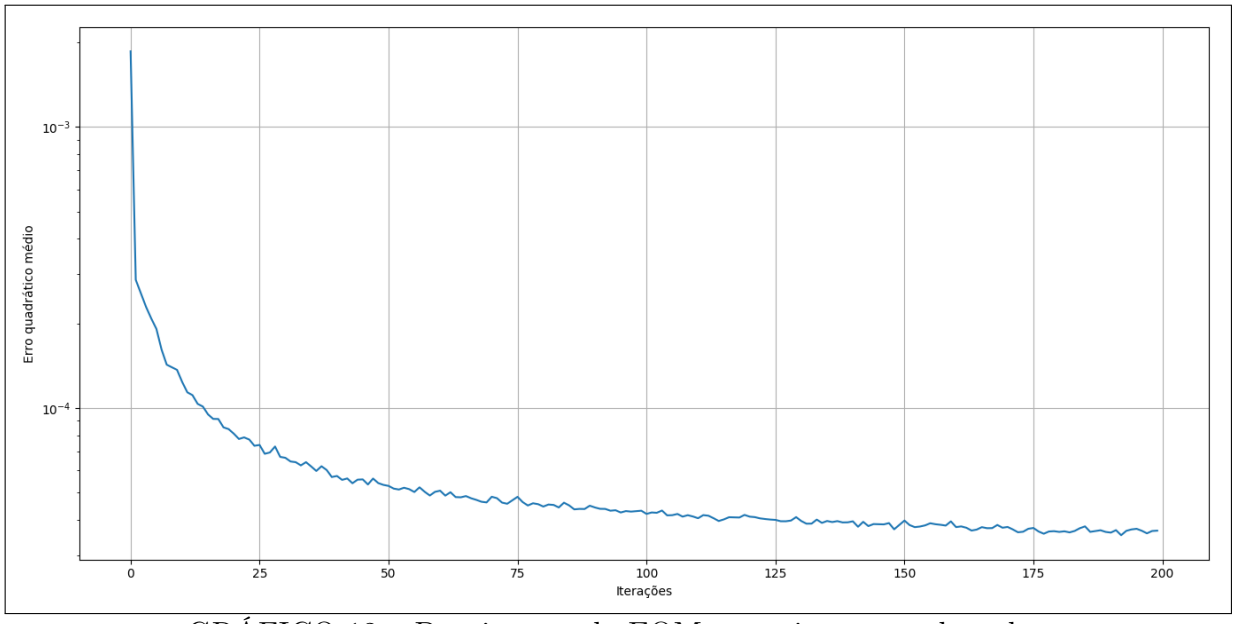


GRÁFICO 12 – Decaimento do EQM no treinamento da rede
FONTE: Elaborado pelo autor

Analisando o gráfico, pode-se observar que os valores tiveram um decaimento constante até a iteração de número 125, enquanto que, nas 75 iterações posteriores, manteve os valores dos erros aproximados. A queda constante nas iterações iniciais está diretamente ligada a capacidade de aprendizado e adaptação ao modelo de dados.

Após a análise do comportamento do EQM e a validação do algortimo de inicialização, os testes foram iniciados com o objetivo de verificar a precisão do modelo. Na Tabela 4 são detalhados os valores do período predito.

TABELA 4 – Período dos dados utilizados para testes: Microsoft Corporation

Data	Abertura	Alta	Baixa	Fechamento	Volume
23/08/2017	72.96	73.15	72.53	72.72	137.665,07
24/08/2017	72.74	72.86	72.07	72.69	170.982,82
25/08/2017	72.86	73.35	72.48	72.82	127.943,01
28/08/2017	73.06	73.09	72.55	72.83	145.697,15
29/08/2017	72.25	73.16	72.05	73.05	114.783,82
30/08/2017	73.01	74.21	72.83	74.01	168.978,01
31/08/2017	74.03	74.96	73.80	74.77	276.528,11

Os dados demonstrados na Tabela 4 foram refinados e normalizados de acordo com os métodos implementados no Capítulo 5. Após a execução deste processo, os mesmos foram inseridos na RNA para ativação. Posteriormente, foi realizada a desnormalização e os resultados apresentados. Na tabela 5 é possível verificar os resultados obtidos pela rede.

TABELA 5 – Resultados da predição realizada nos dados utilizados pela rede

Data	Valor real	Resultado	Erro (%)	Variação
23/08/2017	72.96	72.94	0.027	0.02
24/08/2017	72.74	72.61	0.028	0.13
25/08/2017	72.86	72.50	0.086	0.36
28/08/2017	73.06	72.73	0.028	-0.33
29/08/2017	72.25	72.64	0.695	-0.39
30/08/2017	73.01	72.82	0.201	0.19
31/08/2017	74.03	73.75	0.143	0.28

Analisando a Tabela 5, pode-se observar que os resultados obtidos foram significativos, onde o percentual de erro calculado, através do erro relativo percentual, não ultrapassou a margem 0.70%, se aproximando, consideravelmente, dos valores reais. O erro médio, analisado para todo o período de predição, ficou em torno de 0.17%, o que pode ser considerado baixo levando-se em conta o número de iterações utilizadas para treinar a RNA. Já a medida de dispersão dos erros em torno da média obtida (desvio padrão) foi de aproximadamente 0.23%.

Visualizando os erros obtidos, pode-se evidenciar que não houve grande dispersão no período, com excessão do dia 29/08/2017, que foi o mais alto de toda a série. Este erro foi o mais alto porque houve uma queda grande no valor da ação, se comparado ao padrão que a mesma vinha seguindo. Seguindo esta linha de análise, é preciso destacar que houve um aumento significativo no valor da ação, mais especificamente no dia 31/08/2017, porém, diferentemente do momento de queda, a rede se adaptou a este acrescimento e suavizou o erro. O Gráfico 13 demonstra, de maneira ilustrativa, os resultados obtidos.

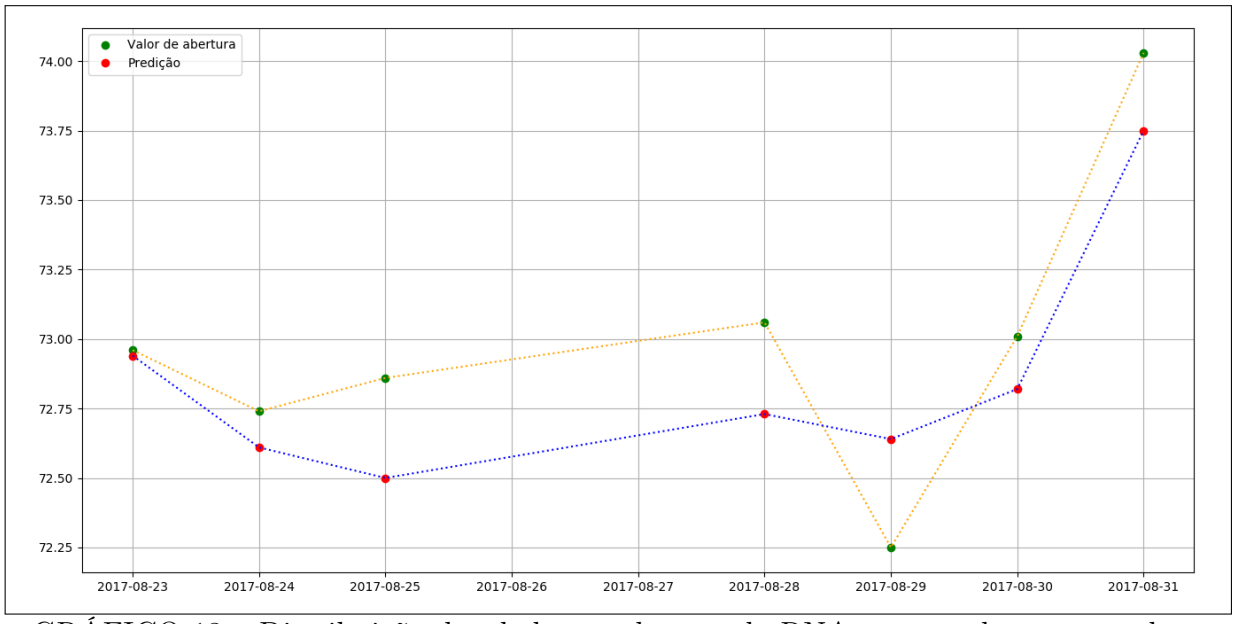


GRÁFICO 13 – Distribuição dos dados resultantes da RNA e seus valores esperados

FONTE: Elaborado pelo autor

A partir destes cenários levantados, é possível chegar a conclusão de que a rede não conseguiu prever com mais exatidão os momentos de queda da ação, enquanto que, nos momentos de alta, obteve uma boa capacidade de adaptabilidade.

6.1.2.2 Treinamento com 1000 Iterações

Com a finalidade de comparar com a simulação anterior, foi realizado um teste com 1000 épocas de treinamento visando alcançar um resultado mais preciso e, também, para validar se o aumento na quantidade de iterações, no processo de treinamento, melhoram os resultados retornados pela rede.

Para quantificar como ocorreu o processo de treinamento, basta multiplicar o número de iterações (1000) com o número de registros de testes (4117), resultando em um total de 4.117.000 exemplos calculados pela rede.

Para o primeiro cenário a rede iniciou-se, na primeira iteração, com um erro percentual de 0.005757041736 e, após o término das 1000 épocas de treinamento, conclui-se com um erro percentual de $3.0266887029028769.10^{-05}$. Mais dois treinos distintos foram realizados visando encontrar alguma dispersão no erro obtido pela rede, como feito na análise anterior. Portanto, o segundo treino obteve um erro inicial de 0.0035660777044 e, após o término das 1000 épocas de treinamento, conclui-se com um percentual de $2.57225977588.10^{-05}$. O terceiro treino obteve um erro inicial de 0.00624617417834 e, após o término das 1000 épocas de treinamento, o EQM do treinamento conclui-se com um percentual de $2.5743685343480545.10^{-05}$. O Gráfico 14 representa, de maneira ilustrativa, a variação do erro resultante no primeiro cenário.

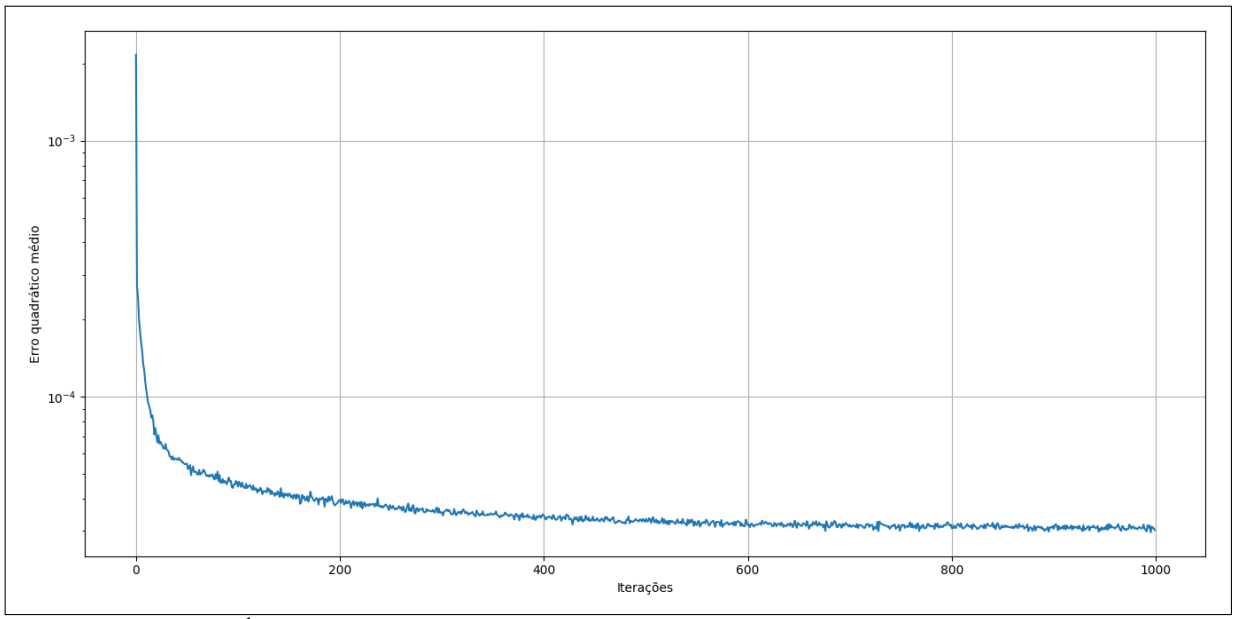


GRÁFICO 14 – Decaimento do EQM no treinamento da rede
FONTE: Elaborado pelo autor

Analisando o Gráfico 14, pode-se observar que o erro obteve uma queda constante

até 600 iterações, enquanto que, entre as iterações de número 600 até a de número 1000, manteve o padrão com valores aproximados, ou seja, foi o momento em que a RNA convergiu. Portanto, após a execução desses cenários de treinamento, pode-se concluir que o modelo se comportou de forma estável em relação a quantidade de iterações, não distinguindo, de forma relevante, os valores dos pesos em diversas inicializações aleatórias.

Após a observação do comportamento do erro, o período de teste (Tabela 4) foi inserido na rede. Os dados foram devidamente refinados, normalizados e, a partir disso, foram ativados. Os resultados preditos, a partir da execução dos testes, são demonstrados na Tabela 6.

REFERÊNCIAS

CARDON, A.; MULLER, N. D. Introdução Às redes neurais artificiais. **Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre**, p. 31, 1994.

CASTRO, M. C. F. **Algoritmo Hebbiano Generalizado para extração dos componentes principais de um conjunto de dados no domínio complexo**. 121 p. Dissertação (Mestrado) — Universidade Católica do Rio Grande do Sul, Porto Alegre, 1996.

CHIZZOTTI, A. Pesquisa em ciências humanas e sociais. **São paulo**, 1991.

CHRISTIE, G. W.; SCHULTZ, H. P. Why do nasdaq market makers avoid odd-eighth quotes. **The journal of Finance**, p. 27, 2003.

CLEMENTS, M. P.; HENDRY, D. F. Forecasting economic processes. **International Journal of Forecasting**, p. 111–131, 1998.

DING, X.; CANU, D.; DENOEUX, T. Neural network based models for forecasting, em proceedings of adt'95. **Wiley and Sons**, 1995.

ELPINIKI, I. P.; KATARZYNA, P. A two-stage model for time series prediction based on fuzzy cognitive maps and neural networks. **Neurocomputing**, 2016.

EXAME. Como funciona a análise fundamentalista de ações. 2010. Acesso em 11 de abril de 2017. Disponível em: <http://exame.abril.com.br/seu-dinheiro/noticias/como-funciona-analise-fundamentalista-acoes-576374>.

FAMA, E. F. Efficient capital markets: A review of theory and empirical work. **The Journal of Finance**, p. 383–417, 1970.

FARIA, E. L. et al. Predicting the brazilian stock market through neural networks and adaptive exponential smoothing methods. **Expert Systems with Applications**, p. 12506–12509, 2009.

FERREIRA, A. A. Comparação de arquiteturas de redes neurais para sistemas de reconhecimento de padrões em narizes artificiais. **Recife**, 2004.

FORTUNA, E. Mercado financeiro: produtos e serviços. **Qualitymark**, 2008.

GAMBOGI, J. A. **Aplicação de redes neurais na tomada de decisão no mercado de ações**. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, 2013.

GEMAN, S.; BIENESTOCK, E.; DOURSAT, R. Neural networks and bias/variance dilemma. **Neural Computing**, p. 1–58, 1992.

GURESEN, E.; KAYAKUTLU, G.; DAIM, U. T. Using artificial neural network models in stock market index prediction. **Expert Systems with Applications**, p. 10389–10397, 2011.

HAYKIN, S. Redes neurais: princípios e prática. **Bookman**, p. 903, 2000.

HAYKIN, S. Neural networks and learning machines. **New Jersey: Prentice Hall**, p. 906, 2009.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing In Science & Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.

JONES, E. et al. SciPy: Open source scientific tools for Python. 2001. Acesso em 10 de setembro de 2017. Disponível em: <http://www.scipy.org>.

KAMIJO, K.; TANIGAWA, T. Stock price pattern recognition a recurrent neural network approach. **Information Technology Research Laboratories, NEC Corporation**, 1990.

KARA, Y.; BOYACIOGLU, A. C.; BAYKAN, O. K. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. **Expert Systems with Applications**, p. 5311–5319, 2011.

KHANDELWAL, I.; RATNADIP, A.; VERMA, G. Time series forecasting using hybrid arima and ann models based on dwt decompositiong. **Procedia Computer Science**, p. 173–179, 2015.

LAVILLE, C.; DIONNE, J. A construção do saber. **Porto Alegre: Artes Médicas, Belo Horizonte: UFMG**, 1999.

LENT, R. Cem bilhões de neurônios? conceitos fundamentais de neurociência. **Rio de Janeiro**, 2001.

LU, W. et al. The modeling of time series based on fuzzy information granules. **Expert Systems with Applications**, p. 3799–3808, 2014.

MARANGONI, P. H. Redes neurais artificiais para previsão de séries temporais no mercado acionário. **Universidade Federal de Santa Catarina, Florianópolis**, 2010.

MCKINNEY, W. **Python for Data Analysis**. [S.l.]: O'Reilly Media, 2013.

MEDEIROS, T. H. **Treinamento de redes neurais artificiais com otimização multi-objetivo e regularização bayesiana: Um estudo comparativo**. 105 p. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, Belo Horizonte, 2004.

MENDONÇA NETO, J. N. **Fractais e redes neurais artificiais aplicados à previsão de retorno de ativos financeiros brasileiros**. 181 p. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, 2014.

NAYAK, S. C.; MISRA, B. B.; BEHERA, H. S. Index prediction with neuro-genetic hybrid network: A comparative analysis of performance. **International conference on computing, communication and applications (ICCCA)**, p. 1–6, 2012.

OLIVEIRA, C. A. Aplicações - redes neurais. **Rio de Janeiro**, p. 1–6, 2002.

OLIVEIRA, M. A. An application of neural networks trained with kalman filter variants (ekf and ukf) to heteroscedastic time series forecasting. **Applied Mathematical Sciences**, p. 3675–3686, 2012.

PANDAS. Powerful python data analysis toolkit. 2017. Acesso em 11 de junho de 2017. Disponível em: <http://pandas.pydata.org/pandas-docs/stable>.

PAOLI, J. et al. Extensible markup language (XML) 1.0 (third edition). 2004. Acesso em 13 de setembro de 2017. Disponível em: <http://www.w3.org/TR/2004/REC-xml-20040204>.

PEREIRA, F. M. Redes neurais artificiais para a predição no mercado acionário brasileiro. **Universidade Federal de Lavas, Minas Gerais**, 2014.

PYCHARM. Docs and demos. 2017. Acesso em 15 de junho de 2017. Disponível em: <https://www.jetbrains.com/pycharm/documentation>.

SCHAUL, T. et al. PyBrain. **Journal of Machine Learning Research**, v. 11, p. 743–746, 2010.

SHILLER, R. J. Irrational exuberance. **New Jersey: Princeton University Press**, p. 296, 2005.

SILVA, J. M. N. Redes neurais artificiais: Rede hopfield e redes estocásticas. **Rio de Janeiro**, 2003.

STANFORD. Convolutional neural networks for visual recognition. 2014. Acesso em 02 de abril de 2017. Disponível em: <http://cs231n.github.io/neural-networks-1>.

TORORADAR. Análise técnica de ações. 2015. Acesso em 10 de abril de 2017. Disponível em: <https://www.tororadar.com.br/investimento/analise-tecnica/analise-tecnica-de-acoes>.

WAIBEL, A. Consonant recognition by modular construction of large phonemic time-delay neural networks. **Denver**, p. 215–223, 1998.

WHITE, H. Economic prediction using neural networks: The case of ibm daily stock returns. **IEEE International on Neural Networks**, p. 451–458, 1988.

ZHANG, G.; PATUWO, E.; HU, M. Forecasting with artificial neural networks: The state of the art. international journal of forecasting. p. 35–62, 1988.

ZáRATE, L. E.; HELMAN, H.; GáLVEZ, J. M. Representação e controle de laminadores tandem baseado em funções de sensibilidade obtidos através de redes neurais. **Sba Controle e Automação**, v. 14, 2003.